Universidade de Aveiro
2007
Departamento de Electrónica, Telecomunicações e Informática

LUÍS FILIPE CAMPOS DE FIGUEIREDO FACEIRA

PLATAFORMA PARA MEDIÇÃO DA QUALIDADE DE SERVIÇO DA OFERTA DE BANDA LARGA EM PORTUGAL

Universidade de Aveiro
2007
Departamento de Electrónica, Telecomunicações e Informática

LUÍS FILIPE CAMPOS DE FIGUEIREDO FACEIRA

PLATAFORMA PARA MEDIÇÃO DA QUALIDADE DE SERVIÇO DA OFERTA DE BANDA LARGA EM PORTUGAL

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Paulo Salvador, Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor Rui Valadas, Professor Associado com Agregação do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri

presidente

Prof. Dr. José Luís Oliveira

professor associado da Universidade de Aveiro

Prof. Dr. Paulo Salvador

professor auxiliar da Universidade de Aveiro

Prof. Dr. Rui Valadas

professor associado da Universidade de Aveiro

Prof. Dr. Susana Sargento

professora auxiliar da Universidade de Aveiro

palavras-chave

Qualidade de serviço, plataforma, web, plataformas, plug-ins, extensibilidade, *auto-updates*, engenharia de software, distribuição aplicacional

Resumo

Existe, hoje em dia, uma oferta muito vasta e diversificada de serviços de acesso à Internet de banda larga. Uma questão que se coloca aos Internautas é a da comparação do serviço prestado pelos diferentes operadores. Este trabalho descreve o desenvolvimento e teste de uma plataforma que permite, por um lado, avaliar em tempo real a qualidade de serviço do acesso à Internet dos seus utilizadores e, por outro, fornecer estatísticas globais da qualidade de serviço oferecida por diferentes operadores, possibilitando a escolha informada do serviço de Internet a contratar. As estimativas de qualidade de serviço são obtidas a partir de medições efectuadas pelos utilizadores da plataforma nos seus pontos de acesso à Internet.

A arquitectura desenvolvida tem como elementos principais um servidor de administração da plataforma e de repositório de dados e uma aplicação de medições periódicas.

O servidor central implementa a base de dados de medições e a interface gráfica principal da plataforma. Nesta interface, é possível a visualização do histórico de medições e a administração dos mais variados parâmetros da plataforma, tais como a lista de tipos de serviço disponíveis, os pontos de medição existentes ou o conjunto de utilizadores registados.

O outro elemento da arquitectura é a aplicação que, após instalação, realiza medições periódicas, sem que seja necessária a interacção dos utilizadores. Esta aplicação possui outras duas funcionalidades de especial destaque: a sua actualização automática e a sua extensibilidade através de plug-ins.

A plataforma encontra-se operacional em ambiente de testes, estando preparada para ser instalada em ambiente de produção.

keywords Quality of service, platform, programming architectures, web, plug-ins, extensibility, auto-updates, software engineering,application deployment

Abstract

Nowadays, there is a wide and varied offer in terms of Internet broadband access. Thus Internet users are required to compare the service provided by the different operators.

This work describes the development and testing of a platform, which, on one hand allows the evaluation in real time of the quality of service of its users Internet access and, on the other hand, provides global statistics related to quality of service provided by the different operators, allowing users to be correctly informed about which Internet Service to choose. The estimations of quality of service are obtained through measurements performed by the platform's users, at their Internet access points.

The main elements of the developed architecture are the platform's administration and data repository server, and an application that performs periodic measurements.

The central server implements the measurements' database and the platform's main graphic interface. In this interface it is possible to visualise the historic of measurements and the administration of the platform's different parameters, such as the list of available Internet services, the existing measurement points or the set of registered users.

The other architecture's element is the application, which, upon installation, performs periodic measurements, without being required the user's interaction. This application has two other important functionalities: its automatic updatability and its extensibility through plug-ins.

The platform is operational in a testing environment and it is ready to be deployed into a production environment.

Índice:

Índice de Figuras

1. Introdução

Actualmente, a oferta de banda larga em Portugal (ADSL, Cabo, *WiFi*) encontra-se disponível de modo generalizado, e o número de utilizadores com este tipo de acesso tem aumentado significativamente. Porém, para realizar comparações entre os vários tipos de serviço disponíveis (nomeadamente entre serviços que oferecem largura de banda idêntica ao cliente), ou para fazer a escolha do serviço que pretende contratar, o utilizador encontra alguma dificuldade em ter acesso a informação sobre a qualidade das ofertas existentes.

Torna-se por isso necessária a existência de uma ferramenta *on-line* que permita avaliar a qualidade de serviço (QdS) da sua ligação de banda larga em tempo real, bem como que apresente a qualidade dos diferentes operadores e contratos de fornecimento de serviços, para que o utilizador possa fazer uma escolha informada.

A ferramenta apresenta também um interesse acrescido para os operadores de telecomunicações que fornecem o serviço de acesso à Internet, pois graças a ela será possível mostrar as vantagens e desvantagens dos diferentes operadores/serviços/contratos.

Assim, uma ferramenta para medição da QdS da oferta de banda larga em Portugal estável e fiável, tem o potencial de se tornar o centro (mediático) de todas as discussões sobre a qualidade do serviço de banda larga em Portugal.

Para possibilitar a sobrevivência de uma aplicação deste tipo, que se pretende que seja usada abundantemente ao longo do tempo, é necessária uma preparação cuidada de duas das funcionalidades que maior longevidade costumam trazer a uma aplicação: a sua actualização automática e a sua expansibilidade.

Este capítulo irá apresentar na sua primeira secção, os objectivos que se pretendiam atingir durante a realização do trabalho realizado no âmbito da presente dissertação.

Na secção Estrutura da Dissertação será descrito a estrutura deste documento, sendo indicado o objectivo e o conteúdo geral que consta em cada capítulo.

1.1. Objectivos

O objectivo deste trabalho é desenvolver uma plataforma *on-line* para medição da qualidade de serviço (QdS) em tempo real da oferta de banda larga em Portugal.

O seu desenvolvimento irá ser centrado nos seguintes componentes:

1. Servidor Central – Deverá existir um servidor de controlo central, no qual será apresentada a ferramenta, se permitirão os testes da qualidade de serviço, se apresentarão os resultados já efectuados da QdS por tecnologia (ADSL, cabo, *Wi-fi* ou satélite), por região geográfica, por operador e por contrato de serviço. A interface

deverá ainda apresentar um fórum de discussão que permita aos utilizadores e operadores dialogar e apresentar as suas dúvidas, reclamações e explicações. Resumindo, o servidor será o "cérebro" de toda a ferramenta, funcionando como o repositório central da informação gerada.

2. Aplicação de Medição – A principal fonte de informação da plataforma deverá ser proveniente de utilizadores que se disponham a instalar uma aplicação residente que efectua análises periódicas à qualidade da sua ligação, remetendo os dados para o servidor central.
3. Infra-Estrutura de Auto-Updates – Para que a evolução da aplicação possa ocorrer de forma sustentada, as actualizações ao software que corre nos utilizadores deverá ser suportada por uma infra-estrutura que automatiza o processo de forma simples e segura. Para tal, deverá ser desenvolvida uma *framework* de Auto-Updates.
4. Infra-Estrutura de *Plug-Ins* – A plataforma deverá estar preparada para ser facilmente extensível. Para tal, deverá ser definida uma infra-estrutura que permita a terceiros programar e distribuir *plug-ins* que se integrem com segurança com a aplicação residente.

O trabalho realizado no âmbito da presente dissertação de mestrado surge como continuação de um projecto de fim de curso da Licenciatura em Engenharia de Computadores e Telemática, realizado no ano lectivo de 2005/2006, pelo mestrando em conjunto com Rúben Correia [cofa2006]. A documentação gerada no ponto de vista de projecto de software (análise de requisitos, modelo de dados, protótipo exploratório), no que diz respeito ao servidor central, é bastante semelhante à previamente gerada no projecto referido, já que os requisitos da plataforma não sofreram alterações de maior.

No âmbito do projecto de fim de curso, foi implementada a totalidade do servidor central, com poucas diferenças para a versão actual, e a primeira versão da aplicação residente, programada em C++.

Durante o presente trabalho de mestrado, foi possível aprofundar a análise de estado da arte em praticamente todos os domínios em que o projecto se pode enquadrar.

A aplicação residente foi totalmente reescrita em código C# sobre a plataforma .NET, tendo-se obtido ganhos de desempenho e de tamanho bastante consideráveis. Esta adaptação foi ainda um requisito para o sucesso das outras duas vertentes sobre as quais o trabalho de mestrado se debruçou: a distribuição automática de actualizações e a expansibilidade por *plug-ins*.

É também objecto da presente dissertação um estudo aprofundado das tecnologias de distribuição aplicacional, quer no que se refere à instalação simples como à actualização automática. Este trabalho culminou com a implementação de uma arquitectura de actualizações automáticas que poderá ser facilmente aplicada a qualquer aplicação, tendo-o sido para a aplicação residente do projecto.

Realizou-se ainda um estudo das tecnologias de expansibilidade, tendo sido produzida uma plataforma de controlo de *plug-ins* que pode ser aplicada à maioria dos projectos de software em tecnologia .NET, tal como o foi para a aplicação da ferramenta de medições periódicas do projecto.

1.2. Estrutura da Dissertação

A presente dissertação de mestrado encontra-se dividida em 11 capítulos principais e um capítulo anexo.

O primeiro capítulo apresenta os objectivos do trabalho e a estrutura da dissertação.

Procede-se então a uma análise do estado da arte nas variadas tecnologias associadas à implementação da plataforma *on-line* de medição da qualidade de serviço, no segundo capítulo. Serão analisadas as tecnologias existentes no que diz respeito a medição de qualidade de serviço, tecnologias de implementação *Web*, plataformas de programação, distribuição aplicacional e *plug-ins*.

O capítulo encerra com uma conclusão e revisão geral do estado da arte.

No terceiro capítulo encontram-se definidos os requisitos do sistema a implementar bem como a fronteira do mesmo, segundo as metodologias comuns de engenharia de software.

No quarto capítulo são definidas as propostas de casos de utilização, do ponto da análise de requisitos do sistema. É apresentada a visão geral, os actores do sistema e a descrição detalhada dos casos de utilização requeridos.

No capítulo intitulado “5. Modelo do Domínio” é apresentado o mapa de conceitos a que deu origem o processo de análise de requisitos da aplicação, mapa esse, que veio a ser utilizado na definição do modelo de dados que suporta a aplicação.

De seguida é apresentado o protótipo exploratório no qual se define a abordagem do projecto de software, identificando-se as tecnologias a utilizar e os conceitos a aplicar. Como resultado da arquitectura candidata, são igualmente apresentados protótipos das interacções na forma de *screenshots* simulados.

O sétimo capítulo descreve a solução final obtida, bem como o decorrer do trabalho, nas 6 principais vertentes do mesmo (*Website*; Applet Java; Aplicação Residente Base; Modelo de Dados; Infra-estrutura de Auto-Updates e Infra-estrutura de *Plug-ins*).

No decorrer do projecto, foi possível coligir variadas apreciações relacionadas com técnicas de medição de largura de banda, pelo que se justificou a escrita do capítulo “7. Considerações sobre técnicas de medição”.

Na oitava parte deste documento são apresentadas conclusões aferidas com a implementação do projecto de software, com destaque para os trechos mais complexos: a aplicação residente, a infra-estrutura de auto-updates e a infra-estrutura de *plug-ins*.

Por último é possível consultar um glossário, uma biblioteca de referências e um extenso anexo onde são documentados os principais casos de utilização implementados.

2. Estado da arte

Actualmente apresentamo-nos numa sociedade cada vez mais informatizada. O acesso à Internet encontra-se bastante difundido, com os acessos a que tipicamente nos referimos como sendo de "banda larga" sendo já considerados como acesso básico.

A oferta de acesso é cada vez mais diversificada e a preocupação com a qualidade de serviço, de forma a permitir, entre outros, o uso de tecnologias multimédia através da Internet, é cada vez maior.

Com o aumento da qualidade disponibilizada, aumentam também os requisitos dos utilizadores, já que surgem diariamente novas aplicações na Internet que necessitam de capacidades de desempenho elevadas.

Também por estas razões, têm havido evoluções nas tecnologias usadas na Web, de grande impacto. Cada vez menos o utilizador se disponibiliza a instalar aplicações residentes, conseguindo cobrir grande parte das suas necessidades com recurso a aplicações que correm directamente no seu navegador Web.

O aumento da complexidade dos sistemas desenvolvidos tem também motivado e tem sido inversamente motivado pelas evoluções que se têm verificado nos ambientes de desenvolvimento e *frameworks* associadas.

Este capítulo irá ilustrar o estado actual das variadas tecnologias que estão associadas ao desenvolvimento da plataforma objecto deste trabalho. Na secção Medição de Qualidade de Serviço é apresentada a evolução actual das tecnologias de medição de qualidade de serviço, desde as técnicas de medição a complexas plataformas distribuídas de medição. Na sequência, será apresentado o estado da arte no que diz respeito a Tecnologias Web. Será ainda documentadas as principais Frameworks de Programação em capítulo próprio, sendo analisadas as plataformas J2SE e .NET. Será ainda analisado em secções próprias o estado da arte no que diz respeito a Distribuição de Aplicações, Plug-ins e Assinatura de Aplicações. Por último serão apresentadas conclusões gerais relativamente ao estado da arte na implementação de sistemas de informação como a que é proposta neste trabalho, com componente Web e componente stand-alone com capacidades de auto-actualização e extensibilidade por *plug-ins*.

2.1. Medição de Qualidade de Serviço

Com a massificação dos acessos à Internet em banda larga, verifica-se cada vez mais a necessidade de aplicações e métodos de medição da qualidade de serviço dos vários fornecedores de Internet.

Nos últimos anos têm surgido diversas ferramentas que permitem analisar a velocidade da ligação de um utilizador.

No entanto, as variáveis que influenciam uma medição são de tal forma diversas que uma medição isolada pouco significado tem para aferir a qualidade de serviço do fornecedor ou mesmo do próprio utilizador. Por esta razão, algumas destas aplicações passaram a registar os valores medidos e alguma informação do utilizador para que seja possível fazer análises com significância estatística.

Nas próximas secções irão ser detalhadas as evoluções que se têm verificado na análise estatística das medições e nos métodos de medição.

2.1.1. Plataformas Distribuídas de Medição

Os resultados de medições individuais ganham relevância quando enquadrados numa quantidade elevada de medições que permita diminuir as interferências de particularidades de alguns utilizadores.

Para resolver esta problemática, algumas aplicações de medição *on-line* evoluíram no sentido de registar os resultados das medições, algumas delas solicitando informações adicionais ao utilizador após a realização dos testes. O problema deste tipo de soluções prende-se com a quantidade de dados recolhida ser geralmente reduzida, já que é necessário que o utilizador realize o teste espontaneamente.

Encontra-se agora a surgir uma nova geração de plataformas distribuídas de medição, na qual o presente trabalho se enquadra. Este tipo de aplicações permitem aos utilizadores instalar uma aplicação no seu computador que arranca automaticamente com o sistema operativo e que realiza medições periódicas, enviando-as ao servidor central. Obtém-se assim medições constantes que permitem tirar conclusões sobre a qualidade de serviço de um fornecedor de Internet, em tempo real.

2.1.2. Métodos de Medição

Uma vez que a maior parte das comunicações realizadas na Internet ocorre sobre os protocolos TCP/IP, também os métodos de medição utilizam, de uma forma geral, estes protocolos. O método mais comum de análise da largura de banda é a de descarregar e carregar de ficheiros para um servidor com capacidades elevadas, medindo o tempo demorado em função da quantidade de informação transmitida. Desta forma, é geralmente analisado o *throughput* TCP, ou mesmo o do protocolo aplicacional HTTP.

Outra medição que é realizada na maior parte das aplicações é a de análise do *round-trip-time*.

Este tipo de técnicas tem essencialmente um grande inconveniente, a permeabilidade da medição ao chamado *cross-traffic*. Uma vez que existem cada vez mais aplicações que enviam constantemente informação através da Internet, a probabilidade de uma medição ser realizada isoladamente sem qualquer interferência, é muito reduzida.

Por essa razão, têm sido desenvolvidas variadas técnicas de medição que pretendem realizar as medições com uma carga de transferência mais reduzida e com uma maior independência do tráfego em circulação no momento da medição.

No âmbito da implementação da aplicação residente de medições periódicas, foram analisadas várias destas técnicas de medição, sendo esta análise alvo de detalhe no capítulo 8.Considerações sobre técnicas de medição.

Têm ainda ocorrido desenvolvimentos nas técnicas de medição de atraso, tendo sido desenvolvidas técnicas que permitem a medição de atrasos num só sentido de medição.

Estas técnicas, implementadas em ferramentas como o J-OWAMP [hveiga2005] (*Java One-Way Active Measurement Protocol*), realizam medições de atraso num só sentido, após sincronização de relógios entre os dois *hosts* intervenientes na medição.

As várias técnicas alternativas de medição encontram-se, de forma geral, em fase de desenvolvimento e investigação, pelo que não são ainda consideradas na produção de plataformas distribuídas de medição.

2.1.3. Métodos de Validação

Uma das questões que se apresenta, na implementação de uma plataforma distribuída de medição de qualidade de serviço, está relacionada com a fiabilidade dos dados introduzidos.

Uma das vantagens das análises de grandes quantidades de informação prende-se com o estabelecimento de correlações estatísticas baseadas em dados introduzidos pelo utilizador, tais como a sua localização geográfica.

Sendo assim, é útil comprovar a veracidade dos dados introduzidos pelos utilizadores. Esta problemática estende-se pelas seguintes vertentes de informação a validar:

1. Fornecedor de Internet;
2. Tipo de Contrato;
3. Localização Geográfica;
4. Resultados da Medição.

Cada uma das referidas vertentes possui formas de controlo distintas.

Para a verificação do Fornecedor de Internet, uma vez que a lista de endereços IP atribuídos a cada ISP é pública, é possível, a grande maior parte das vezes, verificar se o endereço do utilizador se encontra numa rede do operador indicado. No entanto, poderão existir situações concretas nas quais um operador registado na entidade regulamentadora possua mais que uma marca de venda de acesso à Internet, partilhando endereços entre aquilo que, do ponto de vista do grande público, são fornecedores de Internet distintos.

No que diz respeito à validação do tipo de contrato, os operadores não possuem normas globais de atribuição de endereços ou qualquer outro elemento que possa indicar a validade do tipo de contrato. De uma forma geral, a distribuição de ip's efectua-se por duas vertentes, a tecnologia utilizada e a localização geográfica. Mas mesmo nestes elementos, não sendo publicados os intervalos respectivos, torna-se muito dificultada a validação destes.

Ainda que não existam dados públicos de intervalos de endereços atribuídos geograficamente, caso este tipo de organização seja usada no operador, existem técnicas que permitem a sua detecção. As técnicas baseiam-se uma vez mais em informação introduzida pelos utilizadores e análises globais estatísticas. Existem inclusive algumas bases de dados disponíveis *on-line* que contêm este tipo de informação tais como a hostip [hostip] e a geoIP [geoip]. Em ambos os casos é disponibilizada a informação de forma livre, o que é um ponto interessante a seu favor. Porém, rapidamente se verifica que a precisão da localização deixa muito a desejar. Em relação a hostip, o site refere que o grau de correcção se situa próximo dos 55%, enquanto que para o serviço geoIP, é garantida uma taxa de sucesso de 60%, isto para cidades dentro do território dos EUA. Para este último caso, é necessário proceder ao descarregamento de um ficheiro contendo uma base de dados e adaptar as ferramentas oferecidas para obter a informação dessa base de dados, enquanto que no primeiro serviço pode ser usada uma API, acedida através do recurso a um simples método GET. Baseando-se em dados introduzidos também eles pelos utilizadores, sem validação, e na suposição de gamas de endereço estáticas, dentro das redes dos operadores, este tipo de técnicas está condenado ao insucesso.

Por último, é essencial que os resultados da medição não sejam adulterados.

Não existem formas 100% seguras de o impedir, mas genericamente é possível implementar mecanismos de autenticidade da informação, garantindo que foi a aplicação a enviar os resultados e que estes não foram alterados manualmente, com recurso a injectores de pacotes ou outras técnicas semelhantes. Recorre-se habitualmente a técnicas genéricas de encriptação e autenticação de dados, usando mecanismos e algoritmos de segurança, tais como o uso de chaves assimétricas.

Existem habitualmente técnicas adicionais de filtragem das medições, eliminando as medições que produziram dados inconsistentes, tais como resultados de largura de banda superiores à contratada.

De uma forma geral, todas as questões de validação tornam-se pouco relevantes desde que exista uma adesão elevada à ferramenta, pois estatisticamente os dados incorrectos serão difusos perante a quantidade superior de dados correctamente introduzidos.

2.2. Tecnologias Web

Assiste-se actualmente a enormes mudanças na área dos sistemas de informação. As tecnologias que permitem a disponibilização de aplicações via Web têm sofrido bastantes evoluções a par com a maior largura de banda das ligações dos utilizadores.

Estas evoluções estão a causar a migração de muitas aplicações que anteriormente eram aplicações residentes nos computadores pessoais para aplicações distribuídas disponibilizadas a partir do navegador de Internet.

Por outro lado, com as enormes evoluções na interactividade das aplicações Web, assiste-se também a uma enorme consolidação das aplicações e infra-estruturas utilizadas no desenvolvimento dos sistemas Web.

Nesta secção serão descritas algumas dessas tecnologias que são já *standard de facto*.

2.2.1. Páginas Web

Existem várias tecnologias de suporte ao desenvolvimento de páginas web, que oferecem diferentes potencialidades.

Em relação ao conteúdo que uma determinada página apresenta, é normalmente feita uma divisão entre *estático* e *dinâmico*.

No primeiro caso, o desenvolvimento resulta numa solução que, requerendo um esforço menor, não é tão atractiva para o utilizador, visto que existem claras limitações em termos gráficos e de funcionalidades. Páginas estáticas são uma solução válida quando não existem requisitos complexos de funcionalidade ou a necessidade de um aspecto visual mais atraente ou interactivo, e em que o conteúdo apresentado aos diversos utilizadores não necessita de alterações constantes ou personalização (daí a designação de estático).

No segundo caso, os conteúdos disponibilizados pelas páginas são alterados frequentemente, de modo a que o site seja mais atractivo para o utilizador. A forma como é alterada a página pode ocorrer de duas maneiras:

- Páginas contendo conteúdo dinâmico (imagens, texto, formulários…) que pode ser alterado/mudado sem necessidade de recarregar a página. Estas páginas correspondem ao conceito de HTML Dinâmico (DHTML), que inclui o uso de HTML, uma linguagem de *scripting* do lado do cliente (normalmente JavaScript) e uma linguagem de definição de apresentação (por exemplo Cascading Style Sheets – CSS).
- Páginas criadas no momento ("*on-the-fly*") pelo servidor, geralmente baseadas em parâmetros passados no endereço URL ou por formulários HTML. São geralmente usadas linguagens do lado do servidor (*server-side*) para a sua criação, tais como PHP, Perl, ASP, JSP, entre outras. Estas linguagens tipicamente usam o *Common Gateway Interface* (CGI) para produzir os conteúdos dinâmicos. É comum o suporte a interacção com diversos tipos de Sistemas de Gestão de Bases de Dados (SGBD), nomeadamente baseados em SQL (MySQL, PostgreSQL, MS-SQL, etc…)

2.2.2. Servidores Aplicacionais

O mercado de software é cada vez mais abordado do ponto de vista de serviço, ao contrário do habitual paradigma de produto.

Em vez de se pagar pela possibilidade de instalar o *software* produzido no computador local, os custos apenas provêm da utilização, ou são suportados pelo negócio da publicidade.

Esta mudança de paradigma é um dos principais impulsos para a proliferação dos servidores aplicacionais.

Um servidor aplicacional é um motor de *software* que disponibiliza aplicações a computadores cliente ou outros dispositivos. Habitualmente, o servidor aplicacional gere a maior parte, ou mesmo a totalidade, da lógica de negócio e de acesso de dados da aplicação. A principal vantagem deste tipo de arquitecturas é a facilidade gerada no desenvolvimento e manutenção de aplicações, provenientes da centralização do trabalho e do facto das aplicações não necessitarem de ser construídas como um todo, mas em blocos do servidor aplicacional.

O conceito de servidor aplicacional é normalmente associado a gestores de *WebServices* e à infra-estrutura Java J2EE; no entanto, conceptualmente, um servidor Web que possua dinamismo, através de linguagens de programação como o PHP também deve ser entendido como pertencendo a esta classe de servidores.

Na área dos servidores aplicacionais [serverside2005], os mais comuns são o IIS e o Apache. No meio empresarial destacam-se ainda o WebSphere e o Mainframe, este último muito associado a aplicações de *core* bancário.

2.2.3. LAMP

Verifica-se elevada heterogeneidade nas tecnologias utilizadas na implementação de aplicações *Web*. No entanto, há um conjunto de ferramentas que se destaca, sendo considerada uma das combinações mais poderosas, sendo ainda das mais acessíveis economicamente.

Refere-se a plataforma LAMP [kuze98], uma plataforma integrada de cariz *open source*. O acrónimo refere-se ao uso conjunto de:

- **Linux** – sistema operativo,
- **Apache** – servidor *web*,
- **MySQL** – servidor de base de dados;
- **PHP** – linguagem de programação (em alternativa, são também por vezes usadas as linguagens Perl ou Python).

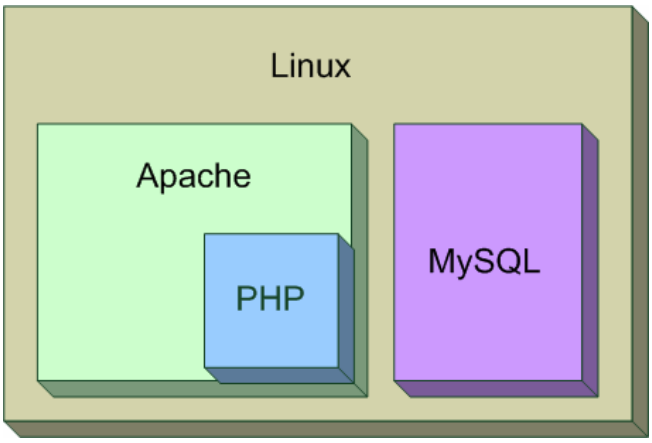


Figura 1 – Arquitectura LAMP

Trata-se de uma plataforma com provas dadas – um caso evidente é o seu uso por parte da maior enciclopédia *on-line* mundial, a Wikipedia [wikipedia].

O termo LAMP foi introduzido por Michael Kunze num artigo [kunze98] numa revista alemã da especialidade. O artigo demonstrava como um conjunto de aplicações gratuitas pode ser uma alternativa viável aos pacotes comerciais que eram, na altura, mais comuns.

O grande rival do LAMP, é o pacote de ferramentas baseadas nos sistemas Microsoft, o WISA [penry2003], Windows, IIS, Sql Server e ASP. No entanto uma vez que o Windows requer

licenciamento, esta alternativa tem perdido bastantes adeptos face ao agora mais popular LAMP.

2.2.4. Tecnologias Client-Side

Numa relação cliente-servidor, as operações realizadas no computador cliente são normalmente referidas como *client-side*.

Sendo cada vez mais abundantes as aplicações cliente-servidor, especialmente com o navegador de Internet como cliente, surge a necessidade de que grande parte do processamento seja realizado pelo cliente, para prevenir sobrecargas dos servidores.

Por outro lado, para aumentar a interactividade das aplicações seria necessária, em alguns casos, uma latência de tal forma reduzida que não é praticável. A solução reside, uma vez mais, em processar as animações e outro tipo de funções que se pretende com resposta muito rápida, no computador do cliente.

No que diz respeito a tecnologias de processamento do lado do cliente, popularizadas, não se pode dizer que haja um consenso na sua utilização.

As tecnologias de processamento *client-side* mais populares actualmente são:

- **Applets Java** – Nos primeiros anos do *boom* da *Web*, o Java surgiu com grande impacto através dos seus Applets Java que permitiam de forma segura executar código complexo no cliente, independentemente do sistema operativo. No entanto, esta tecnologia foi facilmente associada à criação de animações pesadas e desinteressantes nas páginas de cariz pessoal e amador, tendo decaído ao longo dos anos em detrimento do ActiveX e do Flash [adobeflash];
- **ActiveX** – Foi apresentado pela Microsoft como sendo o carrasco do Java, mas é considerado actualmente um grande fiasco. Na prática, os ActiveX são componentes Windows que são executados dentro de um espaço visual de uma página *web*. Por ser uma tecnologia que oferece poucas garantias de segurança aos utilizadores, foi muitas vezes utilizada para auxiliar a proliferação de vírus e outros tipos de código malicioso. Os utilizadores olham com enorme descrédito as aplicações suportadas nesta tecnologia, pelo que o seu uso é cada vez mais reduzido;
- **JavaScript** – Esta linguagem é formalmente denominada de ECMAScript. [ecma262] Normalmente encontra-se associada a pequenas funções de validação ou de alterações ligeiras à interacção dos utilizadores, processada no lado do cliente. No entanto, apesar das primeiras versões desta tecnologia remontam à sua apresentação em 1995 com o navegador de Internet Netscape, é recente a sua ascensão vertiginosa, fruto, principalmente, do aparecimento de uma nova metodologia de desenvolvimento

de páginas *Web* interactivas, o AJAX [garret2005], que se baseia em grande parte na linguagem ECMAScript;

- **Flash** – Esta é a tecnologia que mais desenvolvimento tem sofrido ao longo dos anos. [adobeflash]. Inicialmente apenas era um *plug-in* opcional que permitia realizar animações fluidas e jogos simples. No entanto, com o evoluir da própria tecnologia, e com o aumento da largura de banda dos utilizadores, têm surgido cada vez mais aplicações totalmente centradas nesta tecnologia. O histórico de problemas de segurança deste *plug-in* é muito reduzido e os utilizadores mostram cada vez maior confiança na sua utilização. As maiores reticências no seu uso provêm dos próprios programadores, já que o desenvolvimento desta tecnologia implica uma filosofia de implementação bastante orientada à camada de apresentação, o que é contrário à tendência cada vez maior de entregar esta camada de implementação a artistas gráficos;
- **SilverLight** – Esta é uma nova ferramenta lançada pela Microsoft [silverlight2007] que pretende rivalizar com o Flash, possuindo, de uma forma geral, a mesma filosofia e objectivos. No entanto, ainda é uma tecnologia muito recente, pelo que não se verifica a sua implementação em quantidades consideráveis.

2.3. Frameworks de Programação

As tarefas de desenvolvimento implicam muitas vezes a execução de tarefas repetitivas e a reutilização de código que é aplicável em diversas situações.

Para além disto, a organização interna de uma aplicação ou conjunto de aplicações torna-se cada vez mais importante com o aumento incessante da complexidade dos sistemas desenvolvidos.

Por estas e outras razões, são desenvolvidas diversas estruturas de desenvolvimento e implementação que providenciam soluções pré-codificadas para requisitos comuns das aplicações.

Este tipo de infra-estruturas é habitualmente referido pelo termo inglês *Framework*.

No que diz respeito ao desenvolvimento de aplicações residentes, são duas as principais *Frameworks* utilizadas, a .NET e a Java (J2SE). Nas próximas duas secções serão descritas estas duas alternativas realçando as vantagens e as desvantagens de cada uma.

2.3.1. J2SE

A J2SE (*Java 2 Platform Standard Edition*) [javase] é uma das plataformas de desenvolvimento associadas à linguagem Java [javasun].

Esta plataforma, tal como a linguagem, possui como uma das principais bandeiras de propaganda a sua portabilidade quase total. Existe inclusivamente uma versão simplificada da mesma para dispositivos móveis (J2ME).

De uma forma resumida a J2SE consiste numa máquina virtual (*Java Virtual Machine*) na qual o código Java é executado, e numa vasta biblioteca de métodos e funções habitualmente necessários.

A máquina virtual possui implementações para os mais variados sistemas, implementados pela Sun Microsystems, a empresa criadora da plataforma, bem como de outras empresas de desenvolvimento como a IBM e a Apple.

As normas que definem a linguagem, plataformas ou qualquer componente do universo Java são definidas pela Sun e por uma entidade criada para o efeito denominada Java Community Process. Não existe portanto, qualquer standard suportado por organismos próprios tais como a ECMA (*European Computer Manufacturers Association*), ISO (*International Standards Organization*) ou ANSI (*American National Standards Institute*).

Uma das acusações que o Java sofre regularmente é a de ser relativamente limitado na implementação de aplicações *stand-alone*¸ tal como é visível pela não normalização das bibliotecas de interface, já que são oferecidas duas alternativas não complementares (o AWT e o SWING).

2.3.2. .NET

A *framework* .NET [ecma335] é um componente de software que pode ser adicionado a um sistema operativo, sendo apenas suportado consistentemente em ambiente Windows.

Tal como no J2SE, é fornecido um ambiente de execução controlada e um conjunto de funções e métodos genéricos pré-codificados.

Uma das grandes vantagens desta *framework* é a interoperabilidade através de uma quantidade elevada de linguagens, inclusive suportando o acesso a código que não é executado pela máquina virtual. Tudo isto permite a vários tipos de programadores contribuir com partes de uma solução, com total integração dos módulos desenvolvidos.

As várias linguagens suportadas são codificadas, acrescentadas as funções da *Common Language Infrastructure*, numa linguagem denominada de CIL (*Common Intermediate*

Language) que é depois executada pela máquina de execução do .NET, a *Common Language Runtime*.

O código, ao contrário do Java, não é constantemente interpretado, sofrendo um processo de compilação em tempo de execução pelo compilador JIT (*Just In Time*). Isto confere às aplicações .NET um desempenhoconsideravelmente superior à das aplicações Java.

A infra-estrutura .NET tem sofrido uma enorme evolução, sendo que a cada versão lançada, é iniciado o processo de normalização, sendo o .NET uma norma ECMA [ecma335] e ISO. Isto significa que qualquer pessoa pode implementar a sua própria linguagem na *framework* .NET. Por essa razão, a lista de linguagens que podem correr em .NET é bastante extensa, sendo aqui listadas apenas algumas das principais:

- C# – Uma evolução da linguagem C++, com uma filosofia ainda mais orientada a objectos e mais adequada ao uso da plataforma .NET [ecma334];
- VB.NET – O Visual Basic [microsoftvb] sempre foi uma das principais linguagens de programação em ambiente Windows. Esta é a sua versão para .NET, muito orientada a programadores menos experientes;
- J# – Num esforço de cativar os programadores adeptos da linguagem Java para a *framework* .NET, foi criada esta linguagem [microsoftjsharp] que sucede ao J++;
- PHP – Existe um compilador desenvolvido pela comunidade de software-livre para esta linguagem normalmente usada em ambiente *Web*;
- Ruby – Sendo uma linguagem relativamente recente e que está a crescer exponencialmente, possui já inúmeras implementações em .NET. [matsumoto2000]

No que diz respeito à portabilidade da infra-estrutura para outros sistemas operativos que não da família Windows, a Microsoft implementa apenas soluções para FreeBSD e Mac OS X. No entanto, existem várias iniciativas comunitárias de implementação através de múltiplos sistemas operativos, especialmente para Linux.

2.3.3. Outros

No que diz respeito à implementação de aplicações residentes, independentes de acesso de rede, apenas as plataformas J2SE/J2EE e a .NET se destacam.

No entanto, no que diz respeito a plataformas distribuídas e a desenvolvimento de aplicações *Web*, as hipóteses são imensas.

Serão de seguida descritas as plataformas de desenvolvimento mais populares:

- ColdFusion – Dos mesmos produtores do Flash, este produto [adobecoldfusion] é altamente indicado para programadores de ActionScript e a implementadores muito

orientados ao desenvolvimento gráfico. É utilizado como camada de produtividade ou como camada de serviços numa SOA (*Service Oriented Architechture*) de produção dinâmica de HTML e Flash;

- Struts – É uma plataforma suportada ela própria pelo J2EE, orientada à implementação de sites *Web* dinâmicos. O Struts [holmes2006] extende a API das Servlets de forma a implementar um modelo MVC (*Model-View-Controller*);
- Zend Framework – Plataforma *open-source* [zendframework] para o desenvolvimento de aplicações *Web* em PHP. Disponibiliza uma biblioteca vasta de funcionalidades comuns;
- Ruby On Rails – Esta plataforma [rubyonrails] é uma das que mais tem crescido nos últimos meses, já que é muito orientada às novas tecnologias de interactividade *Web* como o AJAX [garret2005].

2.4. Distribuição de Aplicações

A boa distribuição de uma aplicação é tão importante para o seu sucesso como as funcionalidades e o desempenho global desta.

Por melhor que seja um produto, se não existe uma boa forma de ele ser disponibilizado ao consumidor final, está condenado ao fracasso.

Esta é uma das razões pelas quais as aplicações que correm directamente no navegador *Web* se estão a tornar cada vez mais populares, já que não requerem qualquer tipo de instalação ou configuração, podendo ser executadas de qualquer computador, em qualquer lugar.

Para facilitar a disponibilização de funcionalidades actuais, existem ainda dois mecanismos que são cada vez mais recorrentes, o controlo automático de versões e a expansibilidade por *plug-ins*.

Nesta secção serão discutidas algumas questões associadas ao *deployment* aplicacional, desde os típicos pacotes que geram e facilitam a instalação, a plataformas de disponibilização *on-line*.

2.4.1. Gestores de Instalação, Arranque e Actualização

De todas as questões associadas ao *deployment* aplicacional, aquela que já se encontra mais estudada e consolidada, é a questão da gestão da instalação de aplicações *stand-alone*.

Nos vários sistemas operativos, é comum existir um gestor central das aplicações instaladas e existem diversos produtos que facilitam a criação de pacotes que são inteiramente integrados com estes sistemas. Nesta área, e no que diz respeito ao sistema operativo Windows, há

quatro produtos que possuem um elevado destaque: Microsoft Installer; Wise; InstallAnywhere e o InstallShield. Este último possui inclusivamente versões dedicadas ao *deployment* em ambiente Linux.

Estes gestores de instalação são tipicamente dedicados à distribuição aplicacional *off-line*. No entanto, ultimamente tem surgido uma nova classe de gestores de distribuição, mais orientada ao uso das aplicações *on-line*.

Existem actualmente plataformas que permitem correr aplicações Java ou .NET directamente a partir do clique de um rato numa hiper ligação *Web*.

De notar que estas dificuldades de actualização e distribuição de aplicações ocorrem essencialmente no ambiente pessoal. No que diz respeito a redes empresariais, existem mecanismos bastante evoluídos tais como o SMS (*Systems Management Server*), que controla a realização de *patches*, instalação e desinstalação de aplicações, etc. em redes de várias dimensões.

2.4.2. Java Web Start

São conhecidas as inúmeras limitações que a programação em Java Applets implica. Para ultrapassar as limitações de desenvolvimento em Applets bem como as questões associadas a incompatibilidades geradas pela existência de diferentes máquinas virtuais e *plug-ins* Java dos navegadores Internet, a Sun criou a plataforma Web Start [kim2001].

Esta plataforma permite disponibilizar as típicas aplicações Java possuindo como porta de entrada uma página *Web*. O Java Web Start verifica se a versão da aplicação em *cache* é a mais recente, verifica a necessidade das infra-estruturas de execução, faz o *download* da aplicação e executa-a.

A principal crítica que a comunidade dirige a esta aplicação é o facto de ser necessário que o *plug-in* de Java Web Start esteja, ele próprio, instalado, não resolvendo o problema desde a raiz. Mas, feita justiça, a distribuição de aplicações Java requer incontornavelmente a instalação de *runtimes*, *plug-ins* ou máquinas virtuais e com a maior parte destas, o Java Web Start já vem incluído.

Outro defeito apontado é a falta de integração com os gestores de aplicações dos sistemas operativos, em especial no Windows. Para gerir as aplicações Web Start é necessário recorrer a uma aplicação específica do Java, o que desagrada os utilizadores habituados à centralização deste tipo de informação.

2.4.3. ClickOnce

Os programadores .NET pediam há algum tempo uma metodologia de distribuição semelhante ao Java Web Start. Foi portanto com alguma pompa e circunstância que a Microsoft apresentou, junto com a segunda versão principal da plataforma .NET, a tecnologia ClickOnce [pdarragh2006].

Tal como no Web Start, as aplicações são disponibilizadas directamente numa página *Web*, podendo ser instaladas localmente e executadas *off-line* posteriormente.

Possuem ainda total integração com o gestor de aplicações do Windows.

Para além de simplificar o processo de instalação, esta tecnologia permite adicionar um sistema de controlo de versões às aplicações, já que é possível que esta plataforma verifique a existência de nova versão automaticamente, ao arrancar a aplicação.

O principal atractivo desta tecnologia é a sua facilidade de uso, a sua integração ubíqua com o principal IDE (*Integrated Development Environment*) para .NET, o Visual Studio. Um simples pressionar de um botão e indicação de opções básicas permite disponibilizar uma nova versão de uma aplicação.

No entanto, ainda existem algumas arestas por limar neste produto.

Um problema, muito comum em .NET, é ter sido desenvolvida com o ambiente Microsoft em mente. As aplicações não são correctamente executadas em outros navegadores que não o Microsoft Internet Explorer sem que se encontre instalado um *plug-in* próprio ou sem que sejam realizados alguns passos de configuração avançada.

Outro problema apontado vem associado a limitações de segurança. A instalação de uma aplicação, ainda que não requeira privilégios de administração, apenas é efectuada para o utilizador corrente. Para além disto, os utilizadores recebem pedidos de confirmação de acesso que proporcionam a alguma desconfiança, em parte devido ao histórico dos problemas de segurança do ActiveX.

2.4.4. BITS

BITS é o acrónimo para *Background Intelligent Transfer Service* [mackenzie2004]. O BITS é um serviço disponibilizado nos sistemas operativos Windows, desde o XP, que permite a transferência assíncrona de ficheiros em períodos de inactividade.

É com base neste sistema que algumas aplicações transferem as suas actualizações, já que elas são geridas pelo sistema operativo, não requerendo que a aplicação esteja a funcionar durante a transferência, suportam interrupções na transferência sem que seja necessário reiniciá-la, e não têm impacto na utilização da largura de banda disponível.

O próprio Windows Update, o Serviço de actualizações SMS, ou os mais populares programas de comunicação instantânea utilizam este serviço.

O BITS é disponibilizado através do modelo COM (*Component Object Model*) o que o torna acessível a praticamente todas as linguagens de programação Windows. Por outro lado, não é disponibilizado directamente sobre a plataforma .NET, pelo que o seu uso nas aplicações desta plataforma implica um esforço de configuração e programação de baixo nível a que os programadores .NET não estão muito acostumados.

De notar que o BITS não proporciona directamente mecanismos de actualização automática, sendo apenas uma tecnologia que permite o *download* assíncrono dos ficheiros associados ou não a uma actualização.

2.5. Plug-ins

Especialistas afirmam que se assiste ao iniciar de uma nova era na Internet, a que denominam de *Web* 2.0. Uma das principais características desta nova geração, é a criação comunitária de conteúdo e serviços. Os utilizadores estão cada vez mais participativos e contribuem para o sucesso de uma página ou aplicação.

No que diz respeito a aplicações *stand-alone* o mesmo tipo de situação tem-se verificado na adesão massiva de programadores à criação de *plug-ins* para as variadas aplicações.

A existência de interfaces de programação e extensão em geral, permitem a uma aplicação possuir mais e melhores funcionalidades, com o apoio da comunidade e dos próprios utilizadores. Por esta razão, este tipo de mecanismos tem obtido crescente atenção da parte das produtoras de *software*, em especial as menos preocupadas com o risco de os próprios *plug-ins* se tornarem mais importantes para os utilizadores que a própria aplicação.

No entanto, esta abertura de uma aplicação a possibilidades de interacção com outros pedaços de código criados por terceiros possui quase sempre especificidades próprias da aplicação. Por esta razão não existem plataformas que sejam aceites unanimemente pela comunidade, para o auxílio à implementação de *plug-ins* [jdick2004].

Existem inclusivamente inúmeras formas de expansibilidade muito distintas, desde as alterações visuais (normalmente referidas como *Skins*) a complexos sistemas que tornam a aplicação um simples motor de processamento.

2.6. Assinatura de Aplicações

Com as tecnologias de distribuição e de expansibilidade cada vez mais evoluídas e cada vez mais utilizadas, associam-se questões relativamente complexas de segurança.

No que diz respeito à distribuição, como pode o utilizador certificar-se de que a aplicação é realmente produzida por quem ele pensa que é? Ou que não foi adulterada, introduzido código malicioso?

E quanto à expansibilidade, como garantir que um *plug-in* não vai utilizar as permissões da aplicação principal para realizar operações indesejadas?

Más experiências passadas fazem com que a preocupação com estas questões seja ainda maior. Na informática, não existem soluções de segurança perfeitas, mas os mecanismos de assinatura de aplicações são já bastante evoluídos.

Com a plataforma .NET foi introduzida uma tecnologia denominada Strong Name [seda2004]. Esta tecnologia baseia-se nos consolidados mecanismos de criptografia de chaves assimétricas para assinar zonas de código (intituladas *assemblies*). De uma forma geral, estas zonas são executáveis ou bibliotecas de ligação dinâmica (DLL).

Com este mecanismo é possível, ao carregar dinamicamente pedaços de código, que seja verificada a identidade destes automaticamente pela plataforma .NET, sem que seja necessário implementar quaisquer mecanismos de encriptação ou de *hashing*.

Na distribuição de *updates*, a aplicação principal, antes de aplicar a actualização, pode verificar se esta vem assinada com a mesma chave que a própria aplicação. De notar que todo o tipo de dados, sejam eles código ou não, podem ser assinados com Strong Naming, já que qualquer ficheiro pode ser anexado a um executável ou a uma biblioteca dinâmica.

No que diz respeito a *plug-ins*, em algumas aplicações são definidos diferentes níveis de permissões de interacção em função do facto do *plug-in* ser produzido por um fabricante conhecido ou em função de ter sido ele próprio certificado e assinado pela mesma produtora que a aplicação principal.

De referir ainda que ao usar mecanismos de *hash* nas bibliotecas de código de uma aplicação não se salvaguardam apenas questões associadas à segurança como também questões associadas à gestão de múltiplas versões. O facto de cada código produzir uma assinatura final única, é possível verificar se estamos a carregar código da versão que pensamos estar, resolvendo um enorme problema no desenvolvimento para Windows, conhecido como "*DLL Hell*".

2.7. Conclusão

Como se pode verificar por esta extensa análise do estado da arte em diversas tecnologias de desenvolvimento e implementação de sistemas de informação, poucas são as áreas nas quais existem verdadeiros standards ou tecnologias que são aceites pela totalidade da comunidade.

No entanto, há algumas plataformas, tecnologias ou aplicações que possuem elevado destaque no seu mercado, merecendo referência especial:

- Tecnologias *Web*: Neste campo destacam-se as plataformas LAMP [kuze98] e WISA [penry2003] no que diz respeito a computação no servidor e quanto a computação no lado do cliente, o destaque vai para os Applets Java e para o Flash [adobeflash].
- *Frameworks* de Programação: O .NET e as plataformas Java confrontam-se quase isoladamente pelo domínio da implementação de sistemas, sendo o Java mais orientado a aplicações distribuídas.
- Distribuição de Aplicações: Em gestores de instalação standard, o mais utilizado é o Windows Installer da Microsoft e o InstallShield, para instalação após distribuição manual. Não se encontram ainda suficientemente desenvolvidas as tecnologias de distribuição *on-line*, sendo promissora a tecnologia ClickOnce [pdarragh2006].
- Assinatura de Aplicações: Apenas uma metodologia se destaca que permite a autenticação de pedaços isolados de código, a tecnologia Strong Name da plataforma .NET.

Nas restantes áreas estudadas, não existem plataformas convincentes que justifiquem a sua utilização, já que são limitadas.

A implementação de uma plataforma distribuída de medição de largura de banda não justifica o uso de qualquer biblioteca de validação de dados, de actualização automática, ou de expansibilidade por *plug-ins*, pelo que, se se verificar justificável o esforço, este tipo de funcionalidades deverá ser implementado à medida.

3. Modelo de Requisitos

São apresentados em seguida os requisitos funcionais que foram identificados para o sistema, tendo em vista o conjunto de funcionalidades que se pretendem oferecer aos utilizadores. Ao ser feito o estabelecimento destes requisitos de um modo formal, procura-se que o desenvolvimento posterior dos diversos componentes de software e sua arquitectura estejam de acordo com o definido no início, de modo a que esse desenvolvimento seja focado nas funcionalidades desejadas.

Contudo, dado o processo iterativo decorrente do método aplicado para fazer a análise e desenho do sistema, convém referir que podem ocorrer discrepâncias entre os requisitos aqui definidos (bem como a descrição das narrativas dos Casos de Utilização apresentados no capítulo seguinte) e as funcionalidades apresentadas pelos elementos do sistema, resultantes da natureza dinâmica e evolutiva que se pretende que o sistema tenha. No decorrer do desenvolvimento da plataforma, os requisitos foram inúmeras vezes alterados e actualizados, reflectindo o conhecimento mais profundo do âmbito do sistema, bem como da experiência de utilização feita durante todo o processo.

Na próxima secção serão apresentados os requisitos do sistema e na segunda secção deste capítulo será definida a fronteira do mesmo.

3.1. Requisitos do Sistema

Ref.	*Req. Funcionais*	*Tipo*	*Restrições / Req. Não Funcionais*
1.1	Realizar Medição de LB *On-Line* (applet)	Evidente	Tempo de Medição < 2 minutos Interface: Indicar evolução da medição Medir *download*/*upload*
1.2	Garantir Armazenamento dos Dados de Medição *On-line*	Não Evidente	Dados são “coerentes”
1.3	Verificar consistência entre contrato declarado e endereço IP da medição *on-line*	Não Evidente	Ter *IPranges* detalhados, actualizados e fidedignos.
1.4	Identificar o Utilizador	Não Evidente	Segurança
1.5	Permitir que a aplicação de medição *on-line* seja disponibilizada em diferentes	Não Evidente	Apenas em *mirrors* autorizados

	mirrors		
2.1	Permitir gestão de múltiplos contratos/localizações por utilizador	Não Evidente	
2.2	Permitir a gestão de tipos de contrato e operadores disponíveis no sistema	Evidente	Só o administrador
2.3	Permitir a gestão de contas	Não Evidente	Só o administrador
2.4	Permitir Visualização de Dados Medições, segundo múltiplas perspectivas	Não Evidente	Interface: Tratamento estatístico da informação com o auxílio de gráficos
2.5	Permitir registo de utilizadores no *website*	+ou- Evidente	Verificação de dados
2.6	Limpar Contas Inactivas	Não Evidente	Tempo de Inactividade > 6 meses
2.7	Permitir gestão de conta por parte dos utilizadores	Não evidente	Segurança
3.1	Efectuar Medições Períodicas	Evidente	Período mínimo de 30 minutos, com desfasamento entre utilizadores Medir *download*, *upload*, atraso, jitter
3.2	Garantir Armazenamento de Dados de Medição feitos pela aplicação	Evidente	Filtrar Dados “irrelevantes”
3.3	Garantir incorruptibilidade dos dados das medições	Não Evidente	Segurança
3.4	Verificar Consistência entre contrato declarado e medição	Não Evidente	Ter *IPranges* detalhados, actualizados e fidedignos.
3.5	Aplicação “residente” coloca ícone no systray indicando estado e última medição	Evidente	Ter ícones apelativos e facilmente identificáveis
3.6	Aplicação “residente” permite *log-in* automático	Não evidente	Segurança
3.7	Aplicação “residente” inclui sistema	Não	O update é realizado de forma

	de verificação de versões e eventualmente de auto-update	Evidente	segura
3.8	Aplicação "residente" identifica o utilizador na plataforma	Não Evidente	Segurança
3.9	Detectar automaticamente mudanças entre contracto	Não Evidente	Não ser demasiado incomodativo em pedidos ao utilizador.
3.10	Permitir que o utilizador controle o funcionamento da detecção automática de contractos.	Não Evidente	
3.11	Aplicação permite entrada no sitio web com *login* feito	Não evidente	
4.1	Disponibilizar ajuda para o utilizador do site		
4.2	Gestão/manutenção da ajuda		Administrador
5.1	Disponibilização de fórum de discussão	Evidente	
5.2	Facilidades de gestão do fórum	Evidente	
5.3	Identificação partilhada entre fórum/*website*	Não evidente	
6.1	Disponibilização de notícias	Não evidente	
6.2	Facilidades de gestão das notícias	Não evidente	Administrador

3.2. Fronteira do Sistema

O sistema computacional que se pretende desenvolver consiste numa aplicação "residente" para medições periódicas e um *website* para consulta de estatísticas, gestão de conta(s), medições esporádicas (através da utilização de um Java Applet).

Deverá ser desenvolvida uma estrutura de base-de-dados de suporte ao *website*, permitindo guardar informação relativa a: contas dos diversos utilizadores, suas medições, dados

respeitantes aos fornecedores de acesso Internet e aos diversos contratos/tipos de serviço que disponibilizam, notícias, distritos e concelhos, etc.

O *website* apresentará igualmente fóruns de discussão, que servirão de ponto de encontro para os utilizadores registados trocarem informações e opiniões sobre a temática da banda larga.

Para a parte do fórum, deverá ser escolhida uma solução "pronta a usar", de cariz *open source*, visto que existem inúmeras ferramentas para gestão de fóruns, com capacidades e desempenho elevados. Desta forma, será possível encarar o fórum como um subsistema autónomo, visto que não vai ser desenvolvido código com esse objectivo específico. Procurar-se-á fazer uma integração entre o sistema de autenticação do *website* e do fórum, de modo a ser partilhado, o que representa uma facilidade apreciada por parte dos utilizadores.

A análise de requisitos deste subsistema não será feita, partindo-se do princípio que a solução escolhida apresentará um conjunto de funcionalidades típico, suficiente para o normal funcionamento duma ferramenta deste tipo.

4. Modelo de Casos de Utilização

Na sequência da análise de requisitos, foi possível identificar um conjunto de casos de utilização que possibilitem o cumprimento dos mesmos.

Na secção Visão Geral serão apresentados os diagramas representativos do universo de casos de utilização a implementar, para cada actor. Na secção seguinte serão descritos os actores e por último teremos a secção de Descrição dos Casos de Utilização.

4.1. Visão Geral

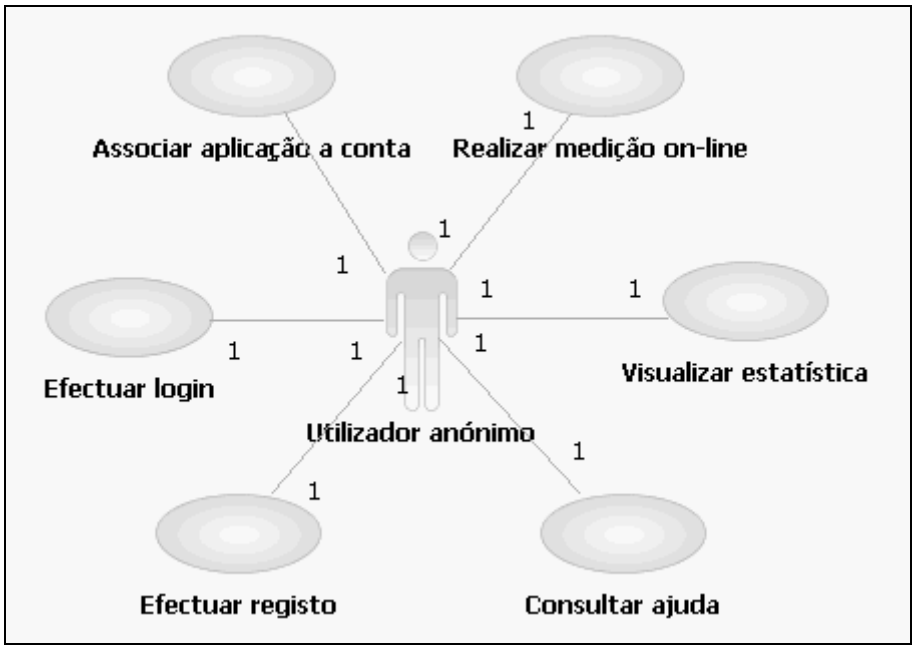


Figura 2 – Diagrama Use Cases do Utilizador Anónimo

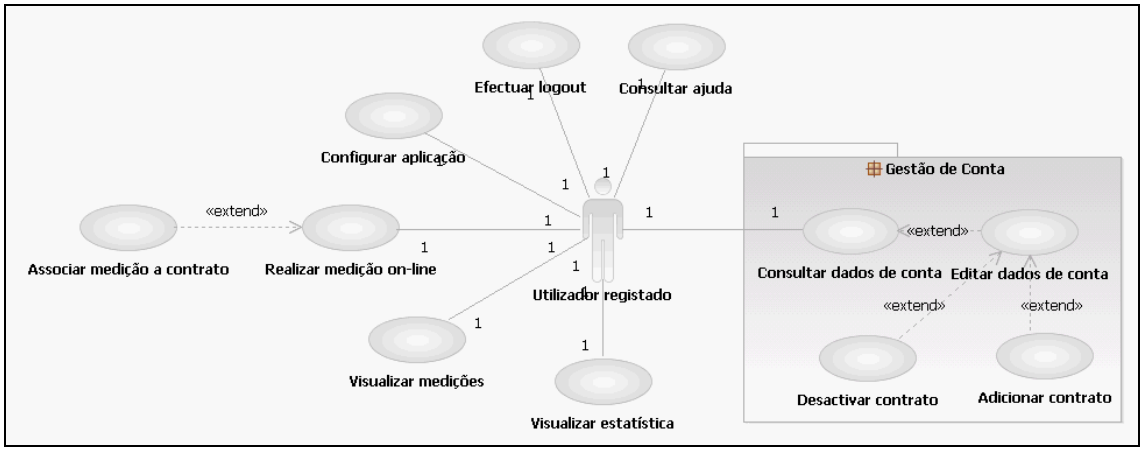


Figura 3 – Diagrama Use Cases do Utilizador Registado

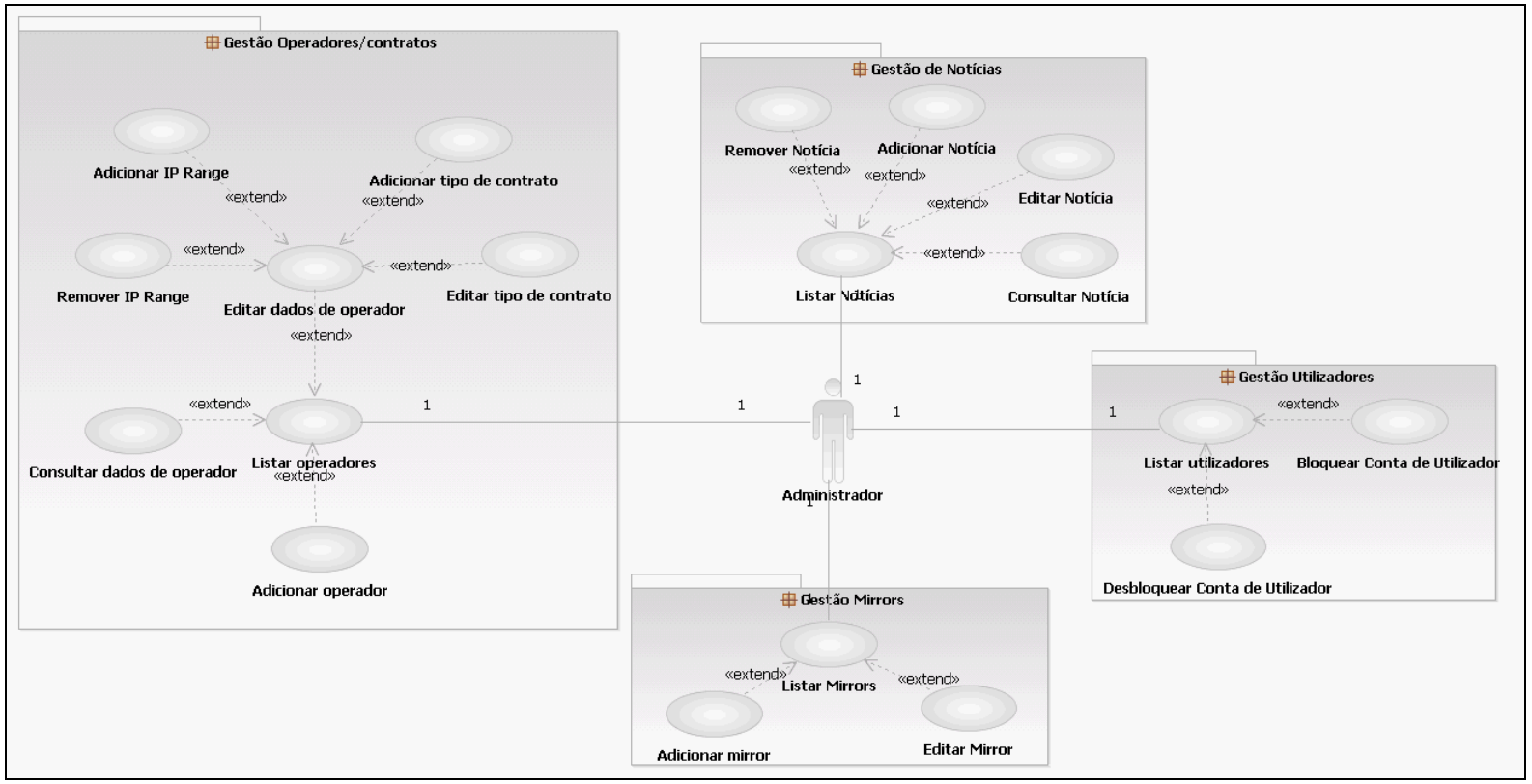


Figura 4 – Diagrama Use Cases do Actor Administrador

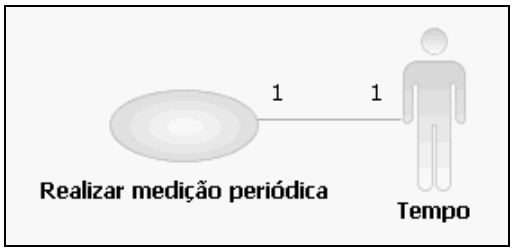


Figura 5 – Diagrama Use Case do Actor Tempo

4.2. Descrição de Actores

Utilizador Registado: entidade que tem uma conta registada, à qual pode fazer alterações, e que está autenticado perante o sistema, podendo desta forma efectuar medições periódicas (através da aplicação residente) ou esporádicas (através da ferramenta *on-line*), passando a ter um histórico das mesmas. Pode visualizar as estatísticas das medições que já efectuou, segundo os contratos que tem registado.

Utilizador Anónimo: entidade que pode consultar a informação estatística geral, referente aos dados globais das medições, e usufruir da ferramenta *on-line* (Java Applet) para efectuar medições. Pode-se registar como utilizador do site.

Administrador: pessoa responsável pela gestão da base de dados, nomeadamente a estabilidade do seu funcionamento e resolução de problemas que possam surgir em "*run-time*", e actualizações pontuais de certos dados.

A gestão do *website*, nomeadamente notícias, *mirrors*, utilizadores, operadores e respectivos tipos de serviço disponibilizados, também é da sua responsabilidade.

Tempo: estímulo periódico que desencadeia acções por parte do sistema (medição periódica pela aplicação residente e verificação da existência de *updates*).

Actores	CaU iniciados
Utilizador Anónimo	Efectuar Registo; Efectuar *Login*; Associar Aplicação a Conta; Realizar medição *on-line*; Visualizar Estatística; Consultar ajuda
Utilizador Registado	Realizar medição *online*; Associar medição a contrato; Configurar aplicação; Efectuar *logout*; Visualizar estatística; Visualizar estatística de Utilizador; Visualizar medições; Consultar dados de conta; Editar dados de conta; Adicionar contrato; Desactivar contrato; Consultar ajuda

Administrador	Listar operadores; Listar utilizadores; Visualizar estatística de Utilizador; Adicionar operador; Consultar dados de operador; Editar dados de operador; Adicionar tipo de contrato; Editar tipo de contrato; Adicionar IP Range; Remover *IP Range*; Bloquear conta de utilizador; Desbloquear conta de utilizador; Listar *Mirrors*; Adicionar *Mirror*; Listar Notícias; Adicionar Notícia; Consultar Notícia; Editar Notícia; Remover Notícia
Tempo	Realizar Medição Periódica;

4.3. Descrição dos Casos de Utilização

Na presente secção, vão ser apresentadas as narrativas estruturadas que correspondem à descrição detalhada dos diversos casos de utilização, anteriormente listados para os actores identificados. Trata-se de uma apresentação e descrição exaustiva de todos os casos de utilização que fazem parte do sistema, com o intuito de dar a conhecer os pormenores de quem e como é realizado. Pretende-se dar uma panorâmica geral do caso de utilização, para se apreender a sua essência, não sendo vinculativa em relação a interfaces em particular.

Nalguns casos particulares, em que seja manifesto o seu interesse, será igualmente apresentado um diagrama de actividade sobre o caso de utilização em questão, com o objectivo de auxiliar na compreensão dos passos inerentes à sua realização.

De notar que os casos de utilização apresentados neste capítulo enquadram-se num trabalho de análise de requisitos, pelo que os casos de utilização reais, aqueles que efectivamente serão implementados, poderão ser distintos.

Para um maior detalhe técnico e formal de cada um dos casos de utilização deverá ser consultado o Anexo A – Modelo de Casos de Utilização, já que na presente secção será apenas apresentada a funcionalidade de cada caso.

Efectuar registo

Finalidade: Criação de uma nova conta no sistema, fornecendo informação pessoal, sendo esta identificada por um nome de utilizador e palavra-chave.

Efectuar *Login*

Finalidade: O actor identifica-se perante o sistema através do seu nome de utilizador e palavra-chave, de forma a poder usufruir dos serviços disponíveis para utilizadores registados.

Efectuar *Logout*

Finalidade: O actor indica ao sistema que pretende terminar a sessão como utilizador registado.

Realizar medição *on-line*

Finalidade: O utilizador anónimo ou o utilizador registado utilizam a ferramenta *on-line* para efectuar uma medição pontual da largura de banda da ligação do utente.

Associar medição a contrato

Finalidade: É feita uma associação entre a medição feita pelo utilizador registado e um dos contratos que possui na sua conta.

Visualizar estatística

Finalidade: O sistema mostra informação tratada estatisticamente de forma gráfica referente às medições, podendo o utilizador escolher entre vários parâmetros de filtragem.

Visualizar medições

Finalidade: O sistema mostra lista com as últimas medições efectuadas pelo utilizador

Consultar dados de conta

Finalidade: O utilizador registado faz a consulta dos dados que constam na sua conta.

Editar dados de conta

Finalidade: O utilizador registado faz a alteração dos dados que constam na sua conta e insere essas alterações no sistema.

Adicionar contrato

Finalidade: O utilizador registado indica ao sistema que pretende associar um novo contrato ao seu registo.

Desactivar contrato

Finalidade: O utilizador registado indica ao sistema que pretende desactivar um contrato, registado para a sua conta.

Configurar aplicação

Finalidade: O utilizador altera as opções de funcionamento da aplicação.

Consultar ajuda

Finalidade: Utilizador consulta página de ajuda

Listar operadores

Finalidade: Sistema mostra lista de operadores actualmente registados.

Adicionar operador

Finalidade: O administrador adiciona ao sistema informação referente a um novo operador.

Consultar dados de operador

Finalidade: Sistema mostra ao administrador dados relativos a determinado operador

Editar dados de operador

Finalidade: Administrador altera os dados relativos a determinado operador

Adicionar tipo de contrato

Finalidade: Administrador adiciona ao sistema um novo tipo de contrato, para que este possa ser escolhido pelos utilizadores.

Editar tipo de contrato

Finalidade: Administrador altera parâmetros que caracterizam determinado contrato.

Adicionar *IP Range*

Finalidade: Administrador associa gama de endereços IP a um determinado operador

Remover *IP Range*

Finalidade: Administrador remove gama de endereços IP a um determinado operador

Listar utilizadores

Finalidade: Sistema mostra lista de utilizadores actualmente registados.

Bloquear conta de utilizador

Finalidade: Conta do utilizador fica com a sinalização de se encontrar bloqueada.

Desbloquear conta de utilizador

Finalidade: Conta do utilizador fica com a sinalização de se encontrar activa.

Listar *mirrors*

Finalidade: Administrador consulta lista com informação referente aos dados dos diversos *mirrors* registados no sistema

Adicionar *mirror*

Finalidade: Administrador adiciona ao sistema um novo *mirror*, para que este possa ser usado pelos utilizadores para realizarem medições *on-line*.

Editar *mirror*

Finalidade: Administrador edita os dados de um determinado *mirror*.

Listar notícias

Finalidade: Sistema mostra lista de notícias actualmente inseridas.

Adicionar notícia

Finalidade: Fazer a inserção de uma nova notícia no sistema.

Consultar notícia

Finalidade: Permite a consulta da notícia pretendida.

Editar notícia

Finalidade: Fazer a alteração dos dados de uma determinada notícia.

Remover notícia

Finalidade: É feita a remoção de uma determinada notícia

Realizar medição periódica

Finalidade: Fazer a realização periódica de uma medição por parte da aplicação residente. De forma a ter estatísticas representativas ao longo de um intervalo de tempo alargado.

5. Modelo do Domínio

O modelo do domínio consiste, por um lado, na identificação dos conceitos que estão por detrás do sistema que pretendemos desenvolver, e por outro, nas relações que se estabelecem entre esses conceitos. Estes conceitos e a forma como se relacionam ou interagem reflectem a lógica que deverá estar por detrás da estrutura de dados a desenvolver.

Para criar o mapa, foram analisados os requisitos definidos anteriormente e procurou-se obter conceitos que estivessem presentes e que se revelassem essenciais para explicar a lógica subjacente às funcionalidades que se pretendem apresentar.

Desta forma chegou-se ao seguinte mapa de conceitos:

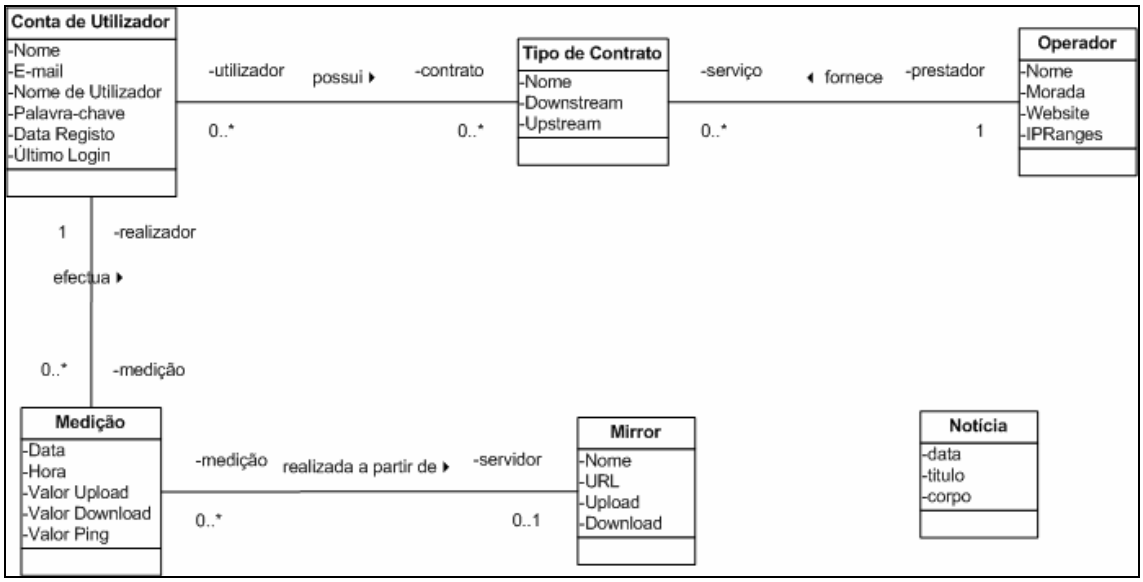


Figura 6 – Mapa de Conceitos

A "Conta de Utilizador" é um conceito que se relaciona com o utilizador registado, servindo principalmente para o identificar perante o sistema. Deverá ter informação de contacto (*e-mail*), um nome que o identifique perante o sistema de forma única (*username*), uma senha para ter acesso à conta (*palavra-chave*), informação temporal relativa ao seu registo (*data registo*) bem como ao seu último movimento (*último login*). Poderá igualmente ter informação relativa ao nome da pessoa.

Visto que um dos objectivos principais de um utilizador registado será o de fazer testes periódicos para aferir da qualidade da sua ligação ao longo do tempo, será necessário que exista algo para suportar isso. A "Medição", que é efectuada por um dado utilizador, deverá ter informação respeitante à *data* e *hora* em que foi feita, bem como os valores dos parâmetros

medidos (*upload*, *download* e "*ping*"). Por *ping* entende-se o atraso de ida-e-volta ou *round trip time* (RTT).

Dado que uma medição pode ser feita a partir de um vários servidores disponibilizados ao utilizador, julga-se ser necessário definir uma entidade que identifique esse servidor, o "*Mirror*", em que conste um *nome* que o identifique facilmente, o endereço *URL* onde se encontra, e eventualmente o valor de *upload*/*download* que o servidor oferece.

Para que seja possível fazer um estudo comparativo entre os vários serviços disponibilizados pelos fornecedores de acesso, é preciso que cada utilizador possa indicar ao sistema os tipos de contrato que possui. O "Tipo de contrato" será identificado pelo *nome*, tendo igualmente informação sobre a velocidade declarada quer para *upload* quer para *download*, e estará ligado ao "Operador" que o fornece. Para esta entidade, deverá existir informação respeitante à sua identificação usual, nomeadamente um *nome* pelo qual é conhecido comercialmente, a *morada* da sua sede social, o endereço do *website*, e uma lista de gamas de endereços IP (*IPRanges*), que servirá para assistir na validação dos registos dos utilizadores.

Por último, foi identificada ainda a necessidade de termos uma "Notícia", que apresenta a *data* em que foi criada (inserida no sistema), um *título* que permita o seu rápido reconhecimento por parte dos utilizadores, assim como o *corpo* da notícia propriamente dita.

6. Protótipo Exploratório

Neste capítulo é apresentada, na primeira secção, a arquitectura candidata de cada um dos componentes que irá ser desenvolvido. Uma segunda secção contém também os Protótipos das Interacções, representações gráficas de como se prevê que seja o aspecto visual das ferramentas a desenvolver.

6.1. Arquitectura Candidata

Esta secção tem por objectivo dar a conhecer as ferramentas que nos vão auxiliar no desenvolvimento dos componentes de software que constituem o nosso sistema, nomeadamente o *website*, o Java Applet e a aplicação residente.

6.1.1. Website

Para o nosso *website*, os conteúdos terão forçosamente de ser dinâmicos e em parte personalizados a cada tipo de utilizador que o use. As funcionalidades disponibilizadas ao utilizador anónimo, ao registado e ao administrador são diferentes, pelo que as páginas também o serão. Deste modo, é essencial adoptar o conceito de páginas dinâmicas, nas suas duas vertentes.

Como iremos ter alguns formulários para preenchimento por parte dos utilizadores, o DHTML é uma boa solução para lidar com esse tipo de elementos, visto que permite fazer uma validação prévia dos dados antes de estes serem enviados para o servidor. Usaremos **CSS** para apresentação dos conteúdos das páginas e **JavaScript** [ecma262] para scripting, principalmente associado à validação de formulários, função para o qual é particularmente apropriado. Estas escolhas prendem-se com o facto de serem duas linguagens consideradas de referência e amplamente usadas, existindo muita documentação, exemplos disponíveis, e já terem sido experimentadas pelos autores do projecto.

O uso de páginas criadas no momento é justificado pelo facto de o sistema ir lidar com uma quantidade significativa de informação que se espera varie frequentemente, como reflexo da realização de medições por parte dos utilizadores, bem como outras alterações de dados menos frequentes (operadores e tipos de contratos, notícias, utilizadores…). Toda esta informação será suportada por uma base de dados e gerida por um SGBD, pelo que é fundamental desenvolver ferramentas em linguagens que permitam, de uma forma ágil, a interacção com esse mesmo SGBD.

Posto isto, foi escolhida uma plataforma integrada de cariz *open source*, vulgarmente conhecida por LAMP [kuze98], acrónimo que se refere ao uso conjunto de:

- **Linux** – sistema operativo,
- **Apache** – servidor *web*,
- **MySQL** – servidor de base de dados;
- **PHP** – linguagem de programação (no nosso caso, visto que também é referido o uso alternativo de Perl ou Python).

Também concorreu para esta escolha o facto de já existir alguma experiência anterior na utilização desta plataforma de desenvolvimento, o que representará uma vantagem em termos de domínio das ferramentas e uma curva de aprendizagem relativamente rápida. Por outro lado, a máquina que foi disponibilizada pelo IT para ser usada como servidor durante o desenvolvimento do projecto correspondia a uma plataforma LAMP, pelo que uma mudança nesse capítulo requereria um esforço adicional, o que não foi considerado como compensatório.

6.1.2. Applet

No que diz respeito à ferramenta de medição *on-line* da largura de banda (uma das principais funcionalidades a disponibilizar ao utilizador), que faz parte do *website*, era preconizado o uso de um Java Applet, como foi definido nos objectivos iniciais do projecto.

Um *applet* é um componente de software (um pequeno programa) que é executado dentro do contexto de outro programa, por exemplo um browser. São usados para fornecer maior interactividade a aplicações *web*, que não seja permitida pelo simples uso de HTML, sendo executadas num ambiente chamado de "*sandbox*", o que restringe as permissões de execução e previne o acesso a dados da máquina local, reforçando assim a segurança para o cliente. O código do *applet* é descarregado de um determinado servidor *web* e o seu interface de utilização é apresentado pelo browser. No caso de *applet* Java, é usado *bytecode* Java, que é independente da plataforma, podendo correr de forma transparente em sistemas Windows, Unix, MacOS, Linux, através do recurso à *Java Virtual Machine* (JVM) ou ao *AppletViewer* da Sun. O facto de ser executado no lado do cliente é positivo, visto que assim alivia carga por parte do servidor, tornando-se assim uma solução que pode ser escalada sem grandes problemas de saturação do servidor.

Contudo, a execução no lado do cliente tem algumas desvantagens, nomeadamente a necessidade de este ter instalado o *plugin* Java, que nem sempre se encontra disponível por omissão em todos os browsers. Além disso, a execução requer que a JVM esteja a correr, o que poderá requerer um tempo de arranque, em certos casos significativo, da primeira vez que é usado o *applet*.

Outra questão prende-se com permissões de acesso à máquina local (do cliente), que geralmente são muito restritas, para garantir maior segurança. Isto pode ser ultrapassado caso o cliente demonstre a sua confiança no autor do *applet*, por exemplo ao aceitar um certificado digital no momento em que o descarrega, podendo o *applet* neste caso ter acesso ao disco e *clipboard* do cliente. Para a ferramenta de medição *online* que será desenvolvida não será necessário ter privilégios de acesso especiais na máquina do cliente, pelo que este problema não deverá ter repercussões no sistema.

Esta tecnologia tem sido diversas vezes usada para o desenvolvimento de pequenas aplicações para fazer medições/testes de largura de banda *online*, tal como é facilmente verificável numa pesquisa rápida pela Internet. A própria referência americana anteriormente mencionada faz uso de um *applet* Java para o teste *online* que disponibiliza aos utilizadores.

Dado que existem muitas soluções para o problema em mãos que fazem recurso ao uso desta tecnologia, e também devido ao facto de já se possuir uma familiaridade razoável com a linguagem de programação Java, considera-se que o uso de *applets* Java se justifica plenamente para desenvolver a aplicação *online* para medição de largura de banda.

6.1.3. Aplicação Residente Base

Em primeiro lugar, foi necessário determinar qual era a plataforma mais vantajosa, neste caso em concreto, de utilizar na implementação da aplicação residente, se a plataforma J2SE ou a .NET. Várias das vantagens e desvantagens de uma e outra foram já detalhadas no capítulo de estado da arte, sendo agora apenas descrito as que mais influenciaram a decisão final, o uso de .NET.

Uma vez que a extensibilidade da aplicação era algo pretendido, foi dada bastante importância à enorme interoperabilidade entre linguagens de programação que a plataforma .NET permite. Facilmente um programador de VisualBasic, de C++, C#, Java, etc. poderá vir a promover o desenvolvimento da aplicação.

Por outro lado, a existência de mecanismos bastante convincentes de segurança no que diz respeito à integração de código autenticado, promoveu também a escolha do .NET.

Outro factor que ajudou na decisão relaciona-se com o desempenho. Para uma boa precisão das medições, o desempenho pode-se revelar bastante importante, e as tecnologias .NET têm demonstrado uma rapidez de processamento superior à execução de código Java.

De referir ainda que o ambiente de programação para .NET, o Visual Studio, é um dos mais utilizados em todo o mundo. As possibilidades fornecidas por este são muitíssimo completas para o Ambiente Windows, o nosso principal alvo de implementação. O facto de possuir uma

comunidade imensamente numerosa proporciona uma facilidade muito maior na resolução de problemas.

O principal argumento que o Java apresentava era o facto de, em teoria, os projectos desenvolvidos possuírem uma portabilidade mais elevada do que as desenvolvidas em .NET. Existem, de facto ambientes de execução Java para muito mais sistemas operativos e processadores do que implementações da CLR do .NET. No entanto, a esmagadora maioria das pessoas utiliza sistemas operativos Windows, pelo que a portabilidade do programa não se considera um aspecto muito relevante, quando em contrapeso com as restantes vantagens da plataforma .NET.

Ambas as tecnologias implicam a presença de um ambiente de execução, mas verifica-se que existe uma maior adesão à plataforma .NET do que à Java, no que diz respeito a aplicações stand-alone.

De referir ainda que a *Java Virtual Machine* limita o uso das suas aplicações a comunicação TCP ou UPD, não permitindo usar, por exemplo, o protocolo ICMP, útil nas medições.

Escolhida a plataforma .NET, restava eleger a linguagem a utilizar.

Uma vez que a aplicação na qual se verificava um maior à-vontade e experiência era a linguagem C++, foi esta a designada inicialmente para a implementação da aplicação.

No entanto, no desenrolar do desenvolvimento, e após o lançamento da primeira versão desta, verificou-se que esta linguagem não se enquadra com facilidade na filosofia de desenvolvimento .NET, pelo que, para tomar o máximo partido desta plataforma, a segunda versão da aplicação já foi totalmente produzida na linguagem C#.

6.1.4. Arquitectura de Auto-Updates

No que diz respeito a arquitecturas [simon2002] que já implementam funcionalidades de controlo de versões, análises periciais não foram muito satisfatórias, pelo que se optou pela sua não utilização.

As aplicações ClickOnce possuem diversas limitações de segurança. Algumas não são ultrapassáveis de todo, e as que são, implicam autorizações especiais dos utilizadores que habitualmente os afugentam.

Uma desvantagem que também desmotivou o uso desta tecnologia foi o facto de que após ser determinado o endereço onde deve ser disponibilizada a aplicação, este não pode ser alterado de forma simples.

Optou-se portanto por um controlo próprio de versões que utilize um formato XML de disponibilização daquilo que normalmente se refere como *Manifest* da aplicação, ou seja, a identificação da versão actual deverá ser publicada em local público, em formato XML, bem como quaisquer dados que sejam guardados temporariamente relativamente ao estado de uma actualização, no computador local.

No que diz respeito à transferência de uma actualização, e já que a aplicação deverá manter um tamanho bastante reduzido, optou-se por implementá-lo directamente, sem recurso à plataforma BITS [mackenzie2004], mais indicada para elevados volumes de transferência.

Todos os ficheiros a ser transferidos deverão ser compilados num único ficheiro dll assinado, para garantir a autenticidade do produto, através de técnicas de Strong Naming [seda2004].

Para que o trabalho realizado possa ser facilmente adaptado para outras aplicações, optou-se pela sua implementação de forma separada e estanque, recorrendo a mecanismos próprios de salvaguarda de informação e configuração.

6.1.5. Arquitectura de Plug-ins

Durante o estudo do estado da arte das arquitecturas de plug-ins, verificou-se que estas são praticamente inexistentes, não existindo nenhuma que pudesse ser satisfatoriamente utilizada no contexto da plataforma distribuída de medição de qualidade de serviço.

Também porque as especificidades de uma arquitectura deste tipo sejam elevadas, verificou-se a possibilidade de implementar mecanismos que permitam a extensibilidade da aplicação segundo três vertentes:

6.1.5.1. Medição

Seria possível permitir a escolha de diferentes métodos de medição, que sejam mais precisos, gastem menos recursos, etc.
Os plug-ins deste tipo poderiam passar um processo opcional de creditação, sendo autenticados como válidos pelos gestores da aplicação, recorrendo aos mecanismos de Strong Naming. Esta creditação permitiria distinguir as medições mais fidedignas das mais duvidosas, para o estabelecimento de análises estatísticas globais.

Os plug-ins poderão ser instalados manualmente, bastando colocá-los numa pasta predefinida. No interface de configuração da aplicação principal, seria possível escolher que método de medição, dentro dos disponíveis, deverá ser utilizado.

O plug-in é carregado dinamicamente, sempre que a aplicação veja necessária a realização de um teste. Sendo assim, as iniciativas de interacção entre a biblioteca de medição e a aplicação deverão ser desta última.

6.1.5.2. Visualização

Não se vê grande utilidade na implementação de mecanismos de *Skinning*, que apenas alteram imagens e cores da aplicação.

O pretendido é que seja possível implementar alternativas a partes ou à totalidade das interfaces da aplicação, funcionando esta como motor de lógica.

Um plug-in destes não requer qualquer tipo de assinatura já que não interfere com a lógica da aplicação, apenas com o aspecto, que é irrelevante para a análise dos dados.

O plug-in deverá expor quais os interfaces que a aplicação deverá deixar de realizar e quais os eventos que pretende que sejam entregues ao plug-in, dentro de uma lista claramente definida.

Uma vez mais, basta que estes plug-ins sejam colocados em pasta própria, e seria possível activar a sua utilização na interface de configuração da aplicação. Deverá ser ainda definido um interface para a solicitação do aparecimento de uma interface de configuração do plug-in, acedido na interface principal de configuração da aplicação.

6.1.5.3. Controlo Total

Uma vez que a aplicação será desenvolvida segundo o paradigma da programação orientada a objectos, será possível disponibilizar individualmente cada funcionalidade, cada interface, cada método, evento ou propriedade a uma entidade computacional autónoma.

Conseguindo isolar-se as diversas componentes da aplicação de forma bastante clara, poderia permitir-se um controlo total da aplicação, passando o plug-in a ser o gestor da aplicação e as diversas componentes ficariam ao serviço do plug-in.

Isto permite que os programadores possam implementar todo o tipo de funcionalidades que não sejam previsíveis, sendo a opção que mais expansibilidade gera.

Este plug-in teria que ser carregado ao iniciar a aplicação, devendo esta ser reiniciada caso se escolha um novo *plug-in* através da interface de configuração da aplicação.

Uma vez que são quase ilimitados os privilégios atribuídos a um plug-in, nesta situação, só serão carregados *plug-ins* que sejam creditados pelos produtores da aplicação com recurso ao mecanismo de Strong Naming.

6.2. Protótipos das Interacções

A presente secção tem por objectivo apresentar os conceitos gráficos idealizados para o nosso sistema, através de um conjunto de diagramas. O desenvolvimento subsequente da solução encontra-se em consonância com o proposto aqui.

Foi definido um conceito, de ***NETeorologia***, que procura reflectir a essência do que é proposto pelo projecto. Visto que o sistema deverá servir como plataforma para realizar testes de qualidade das ligações Internet em banda larga, considera-se interessante fazer uma associação a algo que estivesse relacionado com a medição do estado do tempo (meteorologia), ou mais precisamente, ao *estado da Net* (NETeorologia). Esta ideia surgiu em parte devido a influência da referência francesa, anteriormente mencionada.

Desenvolvendo o conceito, determinou-se que deveria fazer parte integrante do site um mapa, em que constasse Portugal continental e ilhas (ver figura 1), em que fosse mostrado um panorama da qualidade ou estado das ligações de banda larga, em tempo real, através de ícones facilmente compreensíveis.

O ícone atribuído a cada região, que tipicamente engloba um ou mais distritos, reflecte a média da qualidade estimada para o conjunto de medições dessa região. A qualidade corresponde à relação, em termos percentuais, entre o estipulado pelo tipo de contrato que foi associado à medição e o valor efectivamente medido para essa mesma medição.

Além da informação referente ao estado global das várias regiões do País, considerou-se que seria do interesse do utilizador fornecer igualmente o estado da sua ligação, através duma análise estatística idêntica ao acima referido. Esta informação é mostrada no “ecrã” do computador adjacente ao mapa. Considera-se que desta forma é possível disponibilizar informação estatística relevante, com uma interface agradável.

De seguida são mostrados alguns “*screenshots*” que foram criados a partir do desenvolvimento do conceito NETeorologia.

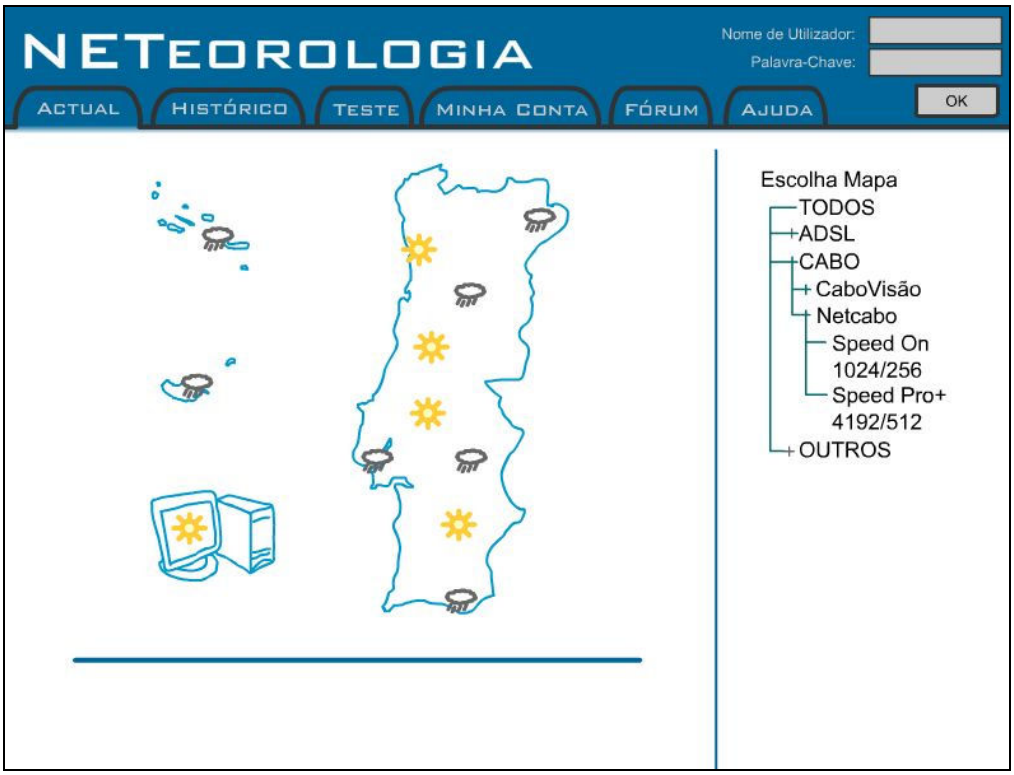


Figura 7 – Página Principal

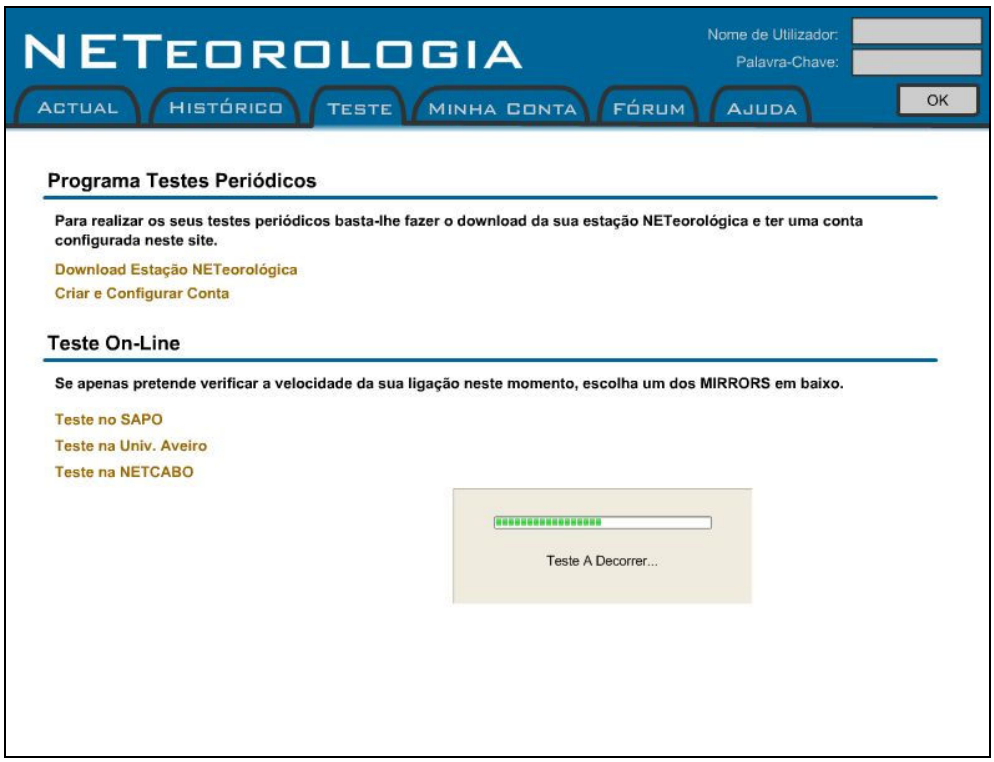


Figura 8 – Teste de *applet*

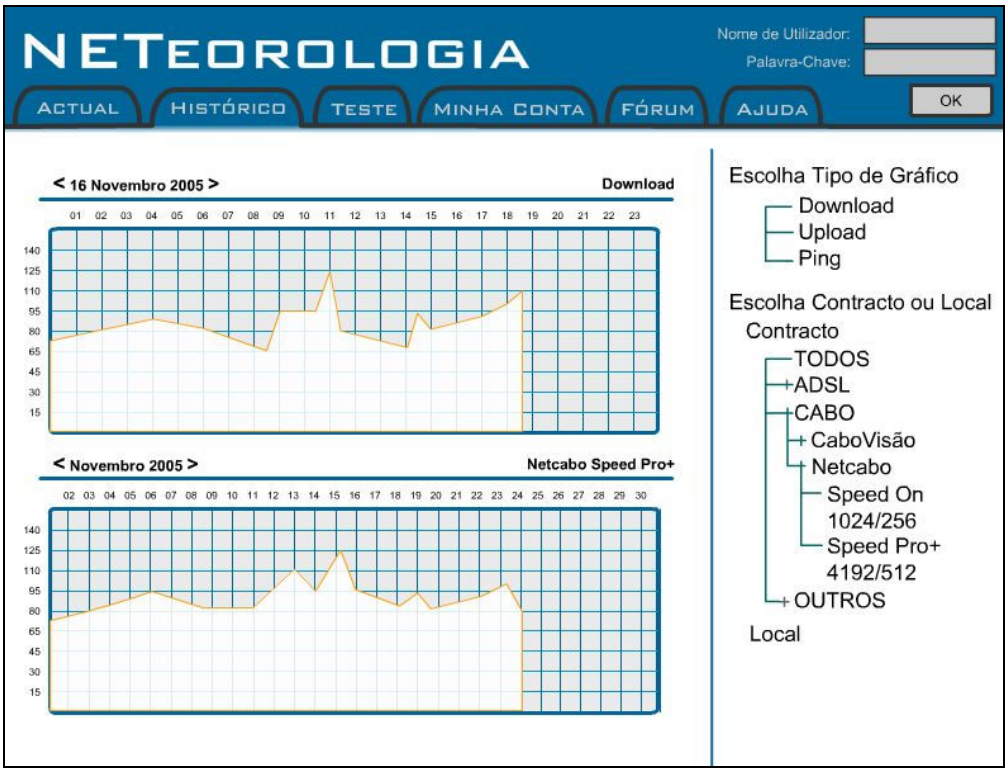


Figura 9 – Página de Histórico

Figura 10 – *Screenshots* da aplicação residente

7. Descrição da solução / implementação

Este capítulo tem por objectivo dar a conhecer a solução de software que foi idealizada e implementada com vista a criar o sistema NETeorologia. As secções dizem respeito a cada uma das várias vertentes de trabalho:

- *Website*;
- Applet de Medição *On-line*;
- Aplicação Residente Base;
- Base de dados de suporte;
- Infra-Estrutura de *Auto-Updates*;
- Infra-Estrutura de *Plug-ins* .

Para o *website*, é feita a apresentação da tecnologia/arquitectura utilizada, bem como os componentes que foram desenvolvidos para criar todo o interface e lógica programática subjacente, com vista a obter toda a funcionalidade anteriormente definida, através dos Casos de Utilização.

Em relação à aplicação residente, é feita uma referência às tecnologias que foram escolhidas para a sua implementação, assim como as classes que foram desenvolvidas e sua interligação…

No que diz respeito ao modelo de dados persistente, ou seja, a estrutura da base de dados que deve ser o repositório de toda a informação do sistema, são apresentadas as relações (tabelas) e respectivas associações que foram definidas, através duma descrição detalhada dos seus respectivos campos, chaves primárias e chaves estrangeiras.

Descrevem-se ainda os métodos e a organização de código, bem como as razões que determinaram a sua criação, no que diz respeito ás infra-estruturas de actualização automática e de extensibilidade por plug-ins.

7.1. *Website*

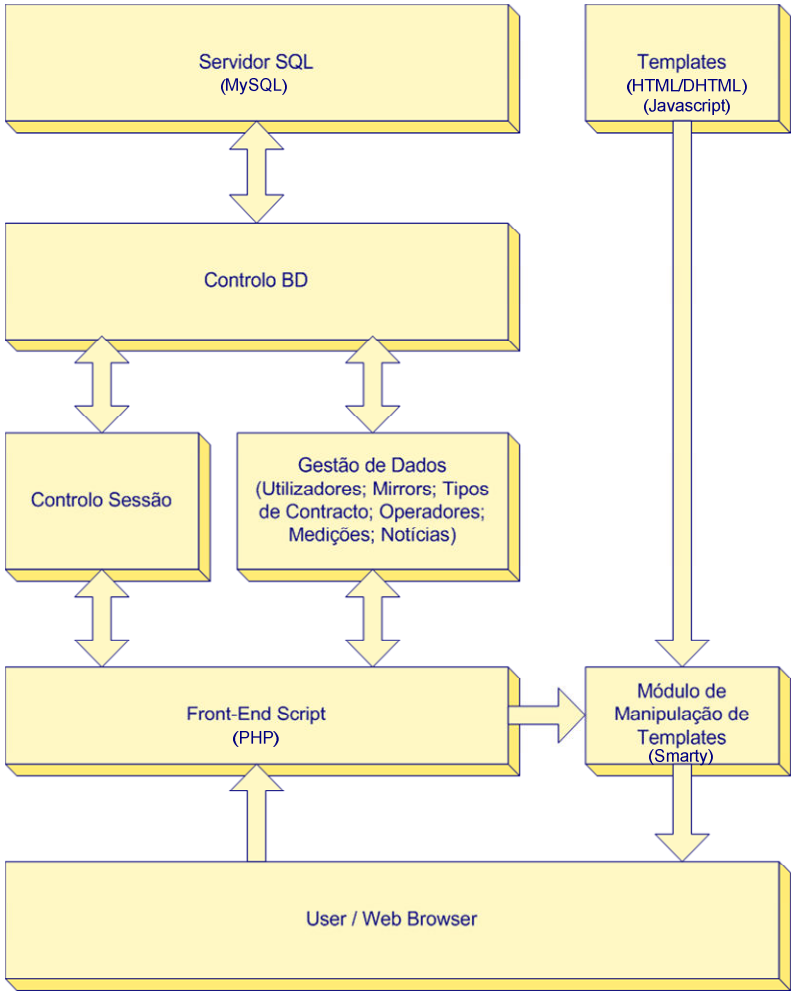


Figura 11 – Estrutura Lógica do *Website*

Para o desenvolvimento do *website*, encontrou-se uma solução que permite obter uma separação clara entre a construção da interface, que será apresentado ao utilizador através das páginas do *website*, e a lógica que serve de suporte à funcionalidade que se pretende oferecer. De um modo geral, procurou-se desta maneira separar o código HTML (bem como CSS e algum código Javascript de apoio) que cria uma estrutura bem definida de apresentação, do código PHP responsável pelas funcionalidades mais complexas que o *website* deve disponibilizar (incluindo a interacção com a base de dados).

Para fazer a interligação entre a camada de apresentação e a camada lógica do *website*, recorreu-se ao uso de um *Template Engine*, tendo sido escolhido no nosso caso o Smarty [smarty]. Trata-se de uma ferramenta que separa o código PHP do código HTML, sendo tipicamente usado para a geração de conteúdos dinâmicos, através da colocação de

"etiquetas" (*tags*) dentro de um documento. Estas *tags* permitem o uso de variáveis (cujo valor é atribuído dinamicamente) e de alguns operadores lógicos e de controlo de ciclos.

Obtém-se assim uma *framework* indicada para o desenvolvimento agilizado de uma aplicação web, visto que a criação de código se torna mais "limpa" e introduz flexibilidade em relação a modificações posteriores. Por exemplo, torna-se trivial alterar a aparência da aplicação (a página web) facilmente sem ser preciso preocupações com a parte de programação lógica "por detrás" da página, pois ela encontra-se separada. De referir ainda que o Smarty oferece um desempenho e escalabilidade consideráveis.

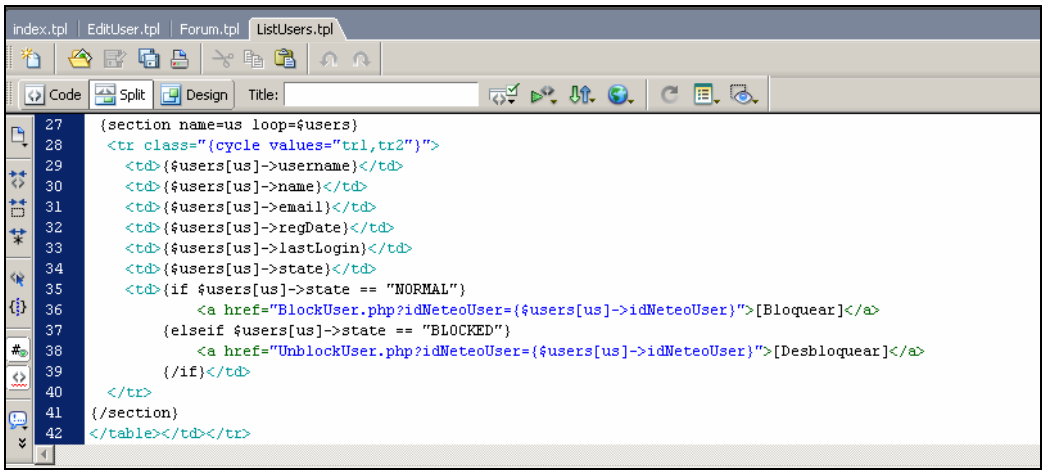


Figura 12 – Exemplo de *template* Smarty para listar utilizadores

Como se pode verificar pela figura anterior, a integração das *tags* faz-se de forma simples dentro do código HTML, permitindo, no exemplo acima, construir uma secção que permite imprimir os valores, passados a partir de um ficheiro PHP, de forma a preencher uma tabela com recurso a um ciclo, agilizando em muito o desenvolvimento das páginas. De notar ainda as *tags* com operadores lógicos (*if/elseif*) que permitem fazer controlo condicional dos conteúdos a mostrar, a partir das variáveis atribuídas através dos ficheiros PHP.

7.1.1. Classes PHP

De seguida é feita uma listagem onde são apresentadas as diversas classes definidas para o sistema, que são responsáveis pela lógica programática que sustenta as funcionalidades do *website*.

Para cada classe são detalhados os métodos que possuem, bem como as variáveis usadas como atributos. Desta maneira, e em complemento com os comentários inseridos no próprio código, pensamos ser possível dar a conhecer em pormenor a forma como construímos a nossa solução de software por detrás do *website*.

Na definição destas classes, procurou-se criar entidades que permitissem por um lado uma interacção mediada com a base de dados e por outro que representassem da forma mais directa possível as relações (tabelas) que a constituem, ou seja, para cada conceito definido (utilizador, operador, tipo de serviço, etc.) ter uma tabela e uma classe que ficasse responsável pelos registos, nomeadamente a sua inserção, actualização, eliminação e listagem.

Daí haver uma relação estreita entre um determinado objecto e o registo que ele deve representar, devendo os valores dos campos do registo estar sempre em consonância com os valores dos atributos correspondentes do objecto.

Considera-se que esta abordagem permite simplificar toda a gestão de registos existentes na base de dados, visto que a divisão por diversas classes, cada uma responsável por parte desses registos, cria uma abstracção que favorece a organização lógica pretendida para o domínio do problema.

Apresenta-se de seguida o objectivo de cada classe implementada, sendo que poderá ser consultado no Anexo B – Classes PHP da Camada Lógica, o detalhe dos métodos implementados.

district.php: Classe responsável pela obtenção de informação referente aos distritos registados na base de dados, comunicando directamente com esta. Possui definido apenas um método

municipality.php: Classe idêntica à anterior (district.php), sendo neste caso responsável pelos concelhos presentes na base de dados.

mirror.php: Classe responsável pela informação referentes aos *mirrors*. Possui 5 atributos (*idMirror*, *baseURL*, *name*, *comments* e *state*), que são inicializados com os valores presentes no registo da base de dados, quando é usado o construtor.

neteouser.php: Classe que tem a responsabilidade pelos registos dos utilizadores. As variáveis que servem de atributo são: *idNeteoUser, email, name, username, pwd, regDate, lastLogin, state* e *trustLevel.*

news.php: Classe que permite gerir os registos das notícias. Possui os atributos *data*, *title* e *body*.

operator.php: Classe responsável pela informação relativa aos operadores. Os atributos que a caracteriza são *idOperator*, *name*, *address*, *website*, *designation* e *state*.

servicetype.php: Classe responsável pela gestão de registos que contêm informação sobre os tipos de serviço disponibilizados por cada operador. Os atributos da classe são *idServiceType*, *refIdOperator*, *name*, *upStream*, *downStream* e *state*.

measurement.php: Classe que tem a responsabilidade de gerir os registos relacionados com as medições. Os atributos que a constituem são *idMeasurement*, *time*, *ip*, *download*, *upload*, *delay*, *source*, *version*, *trusted*, *refIdNeteoUser* e *refIdServiceType*.

session.php – não se trata de uma classe como os casos anteriores, mas antes um módulo de auxílio à gestão das sessões. Como base são utilizadas as facilidades de gestão de sessões embutidas em PHP [php_session], disponíveis a partir da versão 4.0 dessa linguagem de programação.

painter.inc – módulo responsável pela criação da imagem apresentada na página inicial, que consiste no mapa dinâmico com os estados das ligações das diversas regiões do país.

Em termos gerais, o seu funcionamento consiste no seguinte: após criar a imagem base do mapa, para cada uma das doze regiões presentes, obtém o estado actual da qualidade a partir de uma variável global onde estão guardados esses valores; em seguida imprime na imagem base uma imagem correspondente ao estado, nas coordenadas que estão definidas para essa região; no fim de todas as regiões terem sido impressas no mapa, faz a criação da imagem, com o tipo PNG.

Tendo em conta o estado da região do mapa, que consiste num valor entre 0 e 100% ou eventualmente nulo, é escolhida um dos seguintes ícones para impressão:

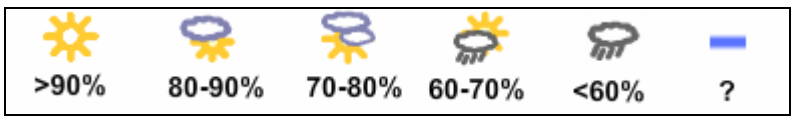


Figura 13 – Representação Gráfica da Qualidade de Serviço

Para estas operações foram usadas funções que fazem parte da biblioteca GD [gd], que permite estender a funcionalidade do PHP de modo a ser possível a criação e manipulação de imagens.

7.1.2. Bibliotecas específicas

ezSQL_mysql – biblioteca [ezSQL] que disponibiliza métodos para fazer a comunicação com o sistema de gestão da base de dados, criando uma abstracção que permite a interacção com esse sistema, aliviando a necessidade de conhecimento de funções específicas. Potencia igualmente a modularidade, pois é possível alterar o SGBD sem que isso afecte o código desenvolvido sobre esta biblioteca, devendo apenas ser adicionado o componente relativo ao SGBD em questão.

Jpgraph – biblioteca para criação de gráficos [jpgraph]

7.1.3. Páginas dinâmicas PHP

Com base nas classes anteriormente referidas, partiu-se para o desenvolvimento da estrutura de apresentação em HTML a partir de *templates*, que têm por função a definição do modo como as páginas são apresentadas e a integração com os dados dinâmicos que são obtidos a partir da interpretação do código PHP associado a cada página.

Após um pedido de uma determinada página por parte do utilizador, o código PHP é interpretado e as variáveis a atribuir ao *template* (através do uso do Smarty) bem como o próprio *template* HTML a usar são definidos em função das condições verificadas (trata-se de um utilizador com sessão activa ou não, o utilizador tem permissão para visualizar a página em questão, os parâmetros foram passados correctamente, etc.). Após a interpretação é então possível criar o código HTML final da página que vai ser enviada de volta ao browser do utilizador.

Serão de seguida listados os ficheiros PHP que implementam a camada de visualização da aplicação, sendo resumido o objectivo de cada uma destas páginas dinâmicas. Para uma compreensão mais detalhada destes ficheiros, deverá ser consultado o Anexo C – *Scripts* PHP da Camada de Visualização.

index.php – página de entrada do *website*. Mostra mapa com a qualidade da banda larga medida nas várias regiões do país, bem como da qualidade da ligação do utilizador, se estiver registado e tiver medições recentes. Apresenta também as notícias actuais.

History.php – esta página mostra os gráficos relativos aos valores de *download*, *upload* e atraso RTT das medições registadas na base de dados, correspondendo ao Caso de Utilização "Visualizar Estatísticas". Permite a aplicação de filtros às estatísticas a mostrar, podendo ser

escolhido o distrito e concelho, o operador e tipo de serviço, a fonte da medição, bem como o dia ou mês que se pretende consultar.

ListMirrors.php – página que permite ao utilizador escolher um *mirror* a partir do qual pode realizar um teste de qualidade da sua ligação, através do *applet*, permitindo iniciar o Caso de Utilização "Realizar medição *online*". Pode igualmente proceder ao descarregamento do instalador da aplicação de medição residente.

EditMirror.php – página que permite a edição de um *mirror* existente no sistema ou o registo de um novo *mirror*, o que corresponde à realização dos Casos de Utilização "Editar *mirror*" e "Adicionar *Mirror*".

InsertMirror.php – página para inserção dos dados de um registo de *mirror* (novo registo ou edição de registo já existente), recebidos por submissão do formulário da página anterior.

EditUser.php – página para registo de um novo utilizador ou consulta/edição da conta de utilizador já registado no sistema. Corresponde respectivamente ao início do Caso de Utilização "Efectuar Registo", "Consultar Dados de conta" ou "Editar dados de conta".

No caso da edição, são ainda disponibilizadas opções que permitem adicionar um novo tipo de contrato, bem como a desactivação de contratos já registados pelo utilizador (concretizando os Casos de Utilização "Adicionar contrato" e "Desactivar contrato"):

InsertUser.php – página de inserção de dados de registo (novo ou edição de um registo já existente), que recebe os dados provenientes do formulário preenchido na página "EditUser.php".

AddContract.php – página que permite fazer a adição de um novo contrato à conta do utilizador, realizando assim o Caso de Utilização "Adicionar contrato". O utilizador define o seu novo contrato através da escolha de um concelho e de um tipo de serviço oferecido por um operador.

Help.php – apresenta uma página de ajuda, com um formato de FAQ.

ListOperators.php – Página responsável pela apresentação dos operadores registados no sistema, sob a forma de uma tabela, correspondendo ao Caso de Utilização "Listar Operadores". São disponibilizadas opções para adicionar um novo registo de operador, bem como para consultar em pormenor ou editar os dados de cada operador listado

(correspondendo a realizar os Casos de Utilização “Adicionar Operador”, “Consultar dados de operador” e “Editar dados de operador”).

EditOperator.php – página que permite adicionar um novo operador, consultar ou editar os dados de um operador já registado no sistema, permitindo a realização dos Casos de Utilização “Adicionar Operador”, “Consultar dados de operador” e “Editar dados de operador”.

InsertOperator.php – esta página tem a função de receber os dados do formulário que consta da página anterior (“EditOperator.php”) e de, após ter feito algumas verificações, os registar na base de dados.

EditServiceType.php – página que permite registar um novo tipo de serviço ou a edição de um que já esteja registado no *website*.

InsertServiceType.php – página responsável pela recepção dos dados respeitantes a um novo tipo de serviço ou edição de um já existente, e sua posterior inserção na base de dados.

AddIP.php – página que permite fazer a adição de uma nova gama de endereços IP, através da definição do endereço de sub rede IP e a máscara de bits. Trata-se da realização do Caso de Utilização “Adicionar IP Range”.

RemoveIP.php – página que permite concretizar a remoção de uma gama de endereços (correspondendo ao Caso de Utilização “Remover IP Range”).

ListUsers.php – página que exibe a listagem de utilizadores registados no sistema, concretizando o Caso de Utilização “Listar Utilizadores”. A listagem pode ser ordenada segundo vários critérios, podendo também ser definido o número de registo a mostrar de cada vez. É possível ainda proceder ao bloqueio/desbloqueio de utilizadores (correspondendo aos Casos de Utilização “Bloquear Utilizador” e “Desbloquear Utilizador”).

BlockUser.php – página responsável pelo bloqueio de um determinado utilizador.

UnblockUser.php – página que permite efectuar o desbloqueio de um utilizador que se encontre bloqueado.

InsertMirrorMeasurement.php – página responsável por receber os dados referentes às medições efectuadas pelos utilizadores, a partir do *applet* disponibilizado pelos *mirrors*, e por obter o valor do atraso RTT (*ping*). Permite ainda ao utilizador autenticado fazer a associação da medição feita a um dos contratos que tem registado no sistema.

ListNews.php – página que apresenta uma listagem com as notícias registadas no sistema, realizando assim o Caso de Utilização "Listar Notícias", mostrando para cada uma a sua data de inserção e o seu título, bem como algumas opções.

EditNews.php – página que permite criar uma nova notícia, consultar ou editar os dados de uma notícia já existente (permitindo a realização dos Casos de Utilização "Adicionar Notícia", "Consultar Notícia" ou "Editar Notícia").

InsertNews.php – está página é responsável pela verificação e inserção dos dados provenientes do formulário da página "EditNews.php".

SACheckAuth.php – esta página permite à aplicação stand-alone verificar a correcção de um par utilizador/palavra-chave. Tal é usado na interface de opções.

SAGetMessages.php – esta página permite à aplicação stand-alone obter mensagens de alerta que poderão ser mostradas ao iniciar a aplicação.

SAGetState.php – esta página permite à aplicação obter o estado do tempo para o utilizador actual, de modo a saber qual o ícone a ser representado na bandeja do sistema.

SAInsertMirrorMeasurement.php – esta página é a que recebe os resultados de uma medição efectuada pelo *Stand-Alone*.

SAListContracts.php – esta página permite ao *Stand-Alone* consultar a lista de contractos que podem ser seleccionados na interface principal

7.2. Applet Java

Um dos componentes de software do sistema é o *applet* Java, que permite aos utilizadores a realização esporádica de testes de largura de banda, através de uma aplicação simples, disponibilizada através de diversos servidores.

Dadas as restrições que esta tecnologia apresenta, de um modo geral relacionadas com questões de segurança, ficou decidido que o *applet* seria responsável pela execução de testes que possibilitassem medir o valor da largura de banda em *download* e em *upload*. Estes são dois parâmetros relevantes para se puder aferir da qualidade de uma ligação em banda larga, visto que, por um lado, se pode quantificar de forma bastante precisa e fazer uma análise comparativa em relação ao contratado pelo cliente ao servidor de acesso Internet, e por outro são dois conceitos com os quais os utilizadores "leigos" destas tecnologias de acesso se encontram bastante familiarizados, dado que são os parâmetros que tipicamente caracterizam uma ligação em banda larga. A sua medição e a divulgação dessa informação ao utilizador através do *applet* representam uma funcionalidade vital do sistema, pelo que o seu desenvolvimento procurou criar uma solução que fosse agradável e simples de usar, quer durante a medição quer na apresentação de resultados, sendo esta última parte atribuída a uma página específica do *website*, (InsertMirrorMeasurement.php).

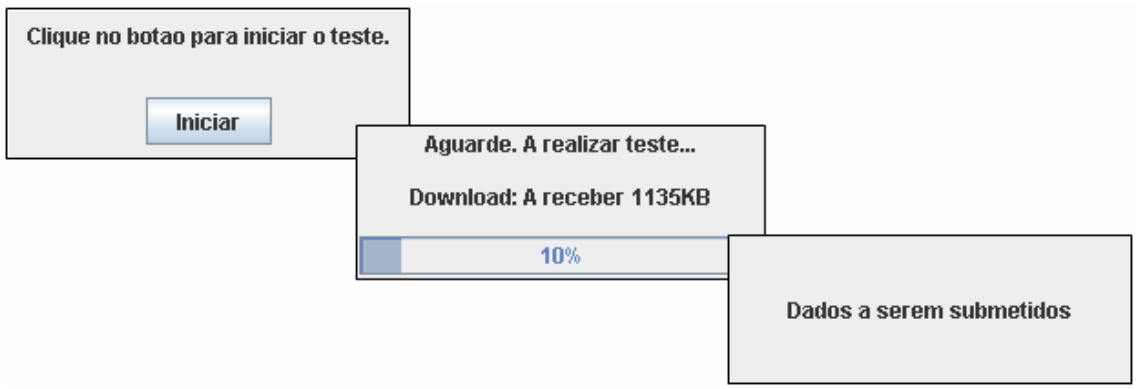


Figura 14 – Sequência de Interfaces do Applet

Como se pode ver pelos *screenshots* da figura 42, o *applet* apresenta um primeiro painel (situado à esquerda na figura), onde é mostrada uma mensagem com instruções para iniciar o teste, bem como o botão "Iniciar", o qual ao ser premido despoleta o início do teste. Após isto, surge um novo painel (na posição intermédia na figura), que mostra informação de forma dinâmica sobre o estado de evolução do teste, através de mensagens de texto, que vão mudando consoante o ficheiro que estiver a ser utilizado, e de uma barra de progresso, que evolui à medida que o ficheiro em causa vai sendo descarregado/carregado (tratando-se respectivamente de *download* ou *upload*).

Para medir o valor disponibilizado em *download*, o *applet* procede ao descarregamento de uma série de ficheiros, com tamanhos crescentes, contendo dados pouco susceptíveis de compressão (no caso usámos ficheiros contendo imagens em formato JPEG). Após o descarregamento de um dado ficheiro, é verificado o tempo decorrido e o número de ficheiros já descarregados. Caso já tenham passado decorrido um determinado intervalo temporal desde o início do teste de *download*, o *applet* não o continua, assim como se já tiveram sido usados

todos os ficheiros, caso contrário descarrega o próximo ficheiro, que possuí um tamanho sensivelmente do dobro do anterior.

A medição do valor de *upload* processa-se de forma similar, mas neste caso são enviados para o servidor, de forma sequencial e com tamanhos crescentes como anteriormente, sequências pseudo-aleatórias de bytes, que as tornam também pouco propícias a sofrerem compressão. Após o envio de cada sequência, são feitas as mesmas verificações de tempo decorrido e existência de mais sequências a enviar.

A verificação do tempo decorrido foi introduzida visto que um dos requisitos definidos para o teste de largura de banda era uma duração limitada no tempo que fosse "aceitável", senão corria-se o risco de o teste demorar um tempo excessivo, o que seria do desagrado do utilizador do sistema. Desta forma, é possível obter algum controlo sobre essa duração, de modo a que o *applet* seja executado dentro de limites que não sejam muito prolongados (os testes feitos revelam que o tempo médio de duração se situa entre os 25-35 segundos).

	500KB	**1MB**	**2MB**	**4MB**	**8MB**
256Kbps	16	32	64	128	256
512Kbps	8	16	32	64	128
2048Kbps	2	4	8	16	32
4096Kbps	1	2	4	8	16
8192Kbps	0,5	1	2	4	8
16384Kbps	<0,5	0,5	1	2	4

A tabela anterior estabelece as relações temporais (em segundos) necessárias para fazer a transferência de um ficheiro de determinado tamanho (500KB – 8MB) usando uma dada largura de banda (256Kbps – 16384Kbps). Os tempos apresentados foram calculados com a fórmula t = L/C, em que t representa o tempo em segundos, L o tamanho do ficheiro em *kilobytes* e a largura de banda em *kilobits/s*. A fórmula não tem em conta *overheads* criados pela técnica de encapsulamento TCP/IP, de modo que estes valores são aproximados, o que é suficiente para o objectivo que se pretendia – ter uma aproximação, ao segundo, do tempo decorrido para a transferência de dados de diversos tamanhos.

Esta tabela foi usada para a definição dos ficheiros – ou mais especificamente, os tamanhos desses ficheiros – que deviam ser disponibilizados para *download*, para que o teste se adaptasse às diferentes larguras de banda que os utilizadores podem ter. Desta forma, o teste deverá ter uma duração idêntica, havendo diferença na quantidade de dados que são descarregados (os tempos a azul indicam qual o último ficheiro que uma determinada ligação

deverá descarregar, se essa ligação disponibilizar o débito máximo para o descarregamento desse ficheiro, tendo em conta a limitação temporal de 10 segundos para os testes).

Após os testes de *download* e de *upload*, são feitas as médias dos vários valores obtidos, sendo estas médias os valores que são passados à página de apresentação dos resultados, para serem inseridos na base de dados. Além destes dois valores, são passados mais alguns, a saber: a versão do *applet* usada (variável *vs*), um valor de controlo *ctl*, uma síntese MD5 [rfc 1321] (variável *md*) e por último o identificador do *mirror* (variável *mr*) em que foi efectuada a medição.

O valor de controlo e a síntese foram acrescentados para fornecer alguma segurança à medição, de modo a evitar ataques em que fossem alterados os valores obtidos pelo *applet*. O controlo é calculado de forma "pseudo-aleatória", servindo como identificador da medição (sendo este o valor inserido no campo de chave primária aquando da inserção de um novo registo de medição na tabela *Measurement*). A síntese é obtida através de uma função que aplica o algoritmo MD5 a uma *string* que resulta da concatenação do valor de controlo com o valor da média de *download*, da média de *upload* e ainda uma sequência de caracteres partilhada entre o *applet* e a página de apresentação de resultados, que funciona como um segredo partilhado (ver diagrama da figura 43).

O uso da sequência de caracteres secreta foi baseado nos mecanismos que são usados para criar *Message Authentication Code* (MAC), que são igualmente conhecidos por "sínteses chaveadas". Desta forma, a página pode verificar se a medição é válida, ou seja, autêntica, fazendo uma operação similar, ao aplicar uma função de síntese MD5 às variáveis que recebe, e comparando com o valor da variável *md*. Caso sejam iguais, trata-se de uma medição que deve ser registada. É assim possível detectar alterações nas variáveis que contêm o valor do *download* e do *upload*, inibindo adulterações desses resultados, visto que, em princípio, apenas o *applet* e a página conseguem gerar sínteses correctas a partir desses valores, devido a partilharem um segredo, apenas conhecido por ambos.

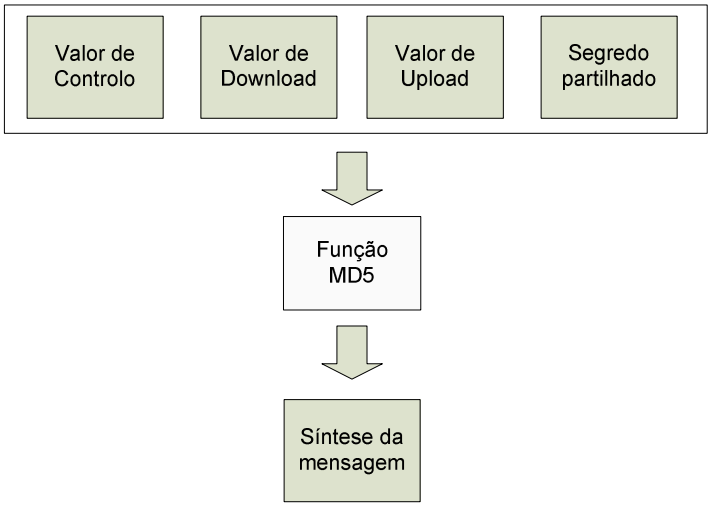


Figura 15 – Diagrama de geração da síntese MD5

De referir ainda que foi escolhido o algoritmo de *hashing* MD5, que pertence à família das funções de síntese. Estas funções produzem um valor de tamanho constante a partir de textos base de tamanho variável, através da aplicação de forma iterativa de função de compressão, de dimensões fixas. Consegue-se assim obter resultados significativamente diferentes para entradas semelhantes (que apresentam poucas mudanças entre si), bem como ser "não invertível", ou seja, dificuldade em conhecer o texto fonte que produziu uma determinada síntese e dificuldade em, dado um texto e sua síntese, descobrir um segundo texto que produza a mesma síntese.

Em relação ao algoritmo usado, trata-se de um dos mais conhecidos e utilizados a nível mundial e apresenta uma significativa robustez em relação a colisões, ou seja, é extremamente improvável obter duas sínteses idênticas a partir de mensagens diferentes. Apesar desta situação poder ocorrer, voluntária ou involuntariamente, a sua probabilidade é ínfima ou exigiria um poder de processamento muito elevado, o que não justificaria de maneira nenhuma o esforço tendo em conta a natureza deste sistema.

Considera-se que usando este algoritmo obtém-se um grau de confiança elevado na solução de segurança idealizada para esta parte do sistema, tendo sempre em mente que o nível de segurança obtido resulta dum compromisso entre as necessidades do sistema em questão e o esforço que se pretende despender no desenvolvimento desses mecanismos de segurança.

7.3. Aplicação Residente Base

Na presente secção vão ser dados a conhecer de forma detalhada os componentes de software que foram desenvolvidos para criar a aplicação residente. Esta aplicação deve permitir ao utilizador, após a sua instalação, a realização de medições periódicas (bem como

de medições esporádicas caso o deseje) de largura de banda em sentido ascendente e descendente, e de atraso médio.

7.3.1. Descrição do Decorrer do Trabalho

O trabalho realizado no âmbito da aplicação residente teve duas fases, das quais resultaram duas versões diferentes.

Será de seguida descrito o decorrer do trabalho ao longo da implementação das duas versões da aplicação.

Em primeiro lugar, para relembrar a linguagem C++ que já não era utilizada há algum tempo, foi programada, de uma forma simples, uma pequena aplicação, executada a partir da linha de comandos, que efectuava o *download* de um ficheiro. Isto possibilitou também uma familiarização com as bibliotecas de comunicação em rede (nomeadamente com a rede Internet), mais concretamente a biblioteca Wininet.

Procedeu-se nesta fase a testes exploratórios da arquitectura de aplicações para Windows MFC. Após ter sido adquirido um domínio razoável dos conceitos inerentes a esta arquitectura, procedeu-se à adaptação da aplicação desenvolvida até então, para que apresentasse uma interface gráfica.

Quando já existia algum conforto na manipulação do projecto MFC, procedemos ao maior salto que o desenvolvimento da aplicação teve. Isolou-se a "inteligência" do teste numa classe própria, criaram-se as interfaces de opções e interface principal, organizando sempre o código com a máxima atenção. Foi então criado o módulo *communicator* para recepção de dados do servidor permitindo assim adicionar a funcionalidade de escolha de contratos.

Após alguns testes e alguma investigação, decidiu-se que as opções seriam guardadas no registo do Windows e foram programados os mecanismos de acesso a este.

Outra questão que se revelou complicada nesta fase foi a da utilização de um ícone na bandeja de sistema e sua manipulação.

Durante estas últimas fases foi sendo necessário adicionar alguns *scripts* de *front-end* ao site *web*, identificados pelo nome que inicia sempre por "SA", para interagir com a aplicação.

Concluídos os principais detalhes prosseguiu-se com a implementação de um instalador para o projecto e a testes de utilização prática.

Foi neste ponto disponibilizada a primeira versão da "Estação NETeorológica".

No entanto, havia diversos aspectos que se gostaria de ver melhorados, na aplicação, causas da falta de experiência na programação em .NET e em C++.

Uma vez que para a segunda fase de implementação, dedicada à extensibilidade da mesma, o uso de tecnologias .NET era fundamental, e já que se verificou que na prática, a primeira fase tinha sido implementada quase sem dar uso a esta plataforma, optou-se por re-implementar a aplicação na linguagem C#.

Sendo o processo de desenvolvimento de software, iterativo, e também porque a linguagem C# e a plataforma .NET a tal o permitem, foram variadas as evoluções da aplicação no que diz respeito a rapidez de execução, gestão de memória, tamanho e organização do código.

Optou-se desta vez pela criação dos métodos globais na classe que representa o próprio programa, já que são implementações específicas do programa, e não serão importadas para outras aplicações.

Foram também melhorados diversos erros da versão anterior e melhorados aspectos de segurança.

Tendo sido desenvolvida entretanto a arquitectura de actualizações automáticas e esta aplicada à aplicação, deu-se origem à publicação de uma nova versão desta.

Alguns erros detectados foram posteriormente corrigidos com recurso à própria infra-estrutura de auto-actualizações, bem como a distribuição da versão que já possui extensibilidade por *plug-ins*.

7.3.2. Arquitectura Final

A aplicação relevou algumas mudanças significativas durante as várias fases de implementação, em especial entre a primeira e a segunda versão da mesma.

A complexidade da arquitectura interna reflecte necessariamente dois aspectos: as características da linguagem e da plataforma bem como a organização do código.

Ao ganhar maturidade, a aplicação simplificou a sua arquitectura, tal como é visível pelos dois diagramas que se seguem que representam a arquitectura da aplicação na primeira e na segunda versão respectivamente.

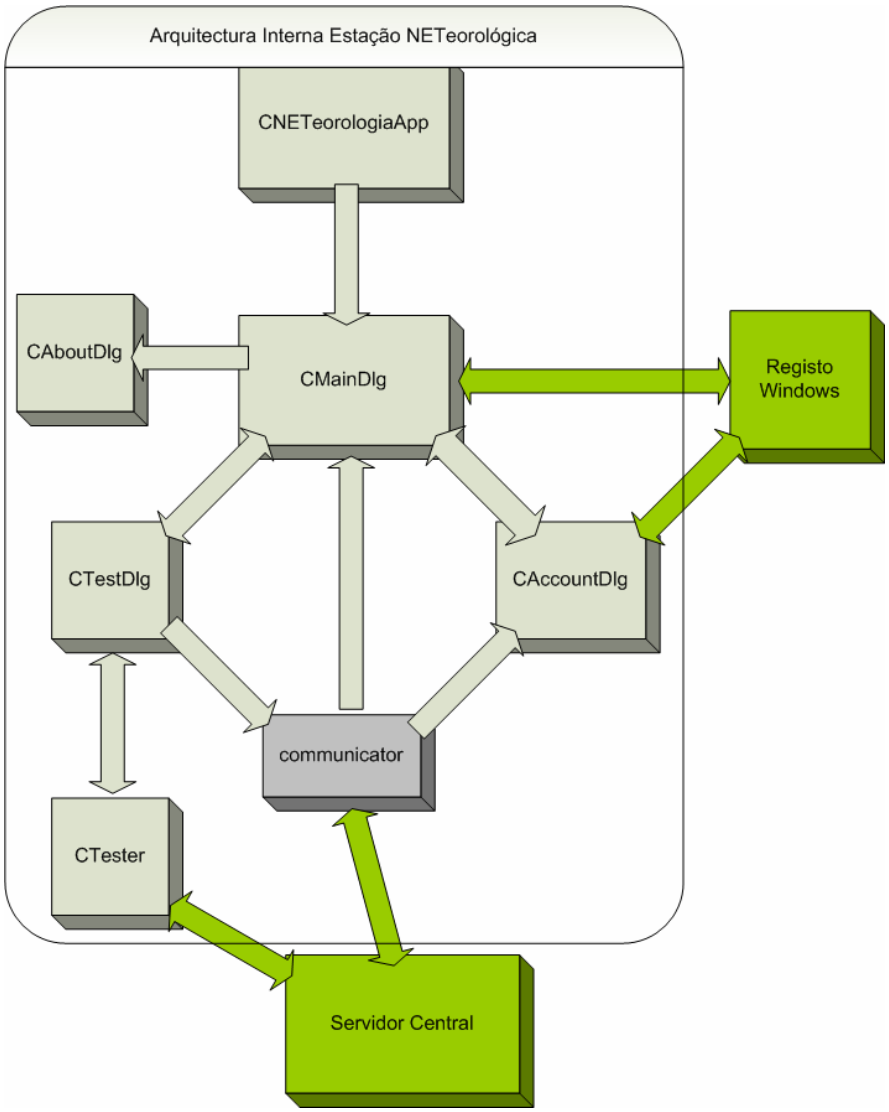


Figura 16 – Arquitectura interna da Aplicação Residente (*StandAlone*)

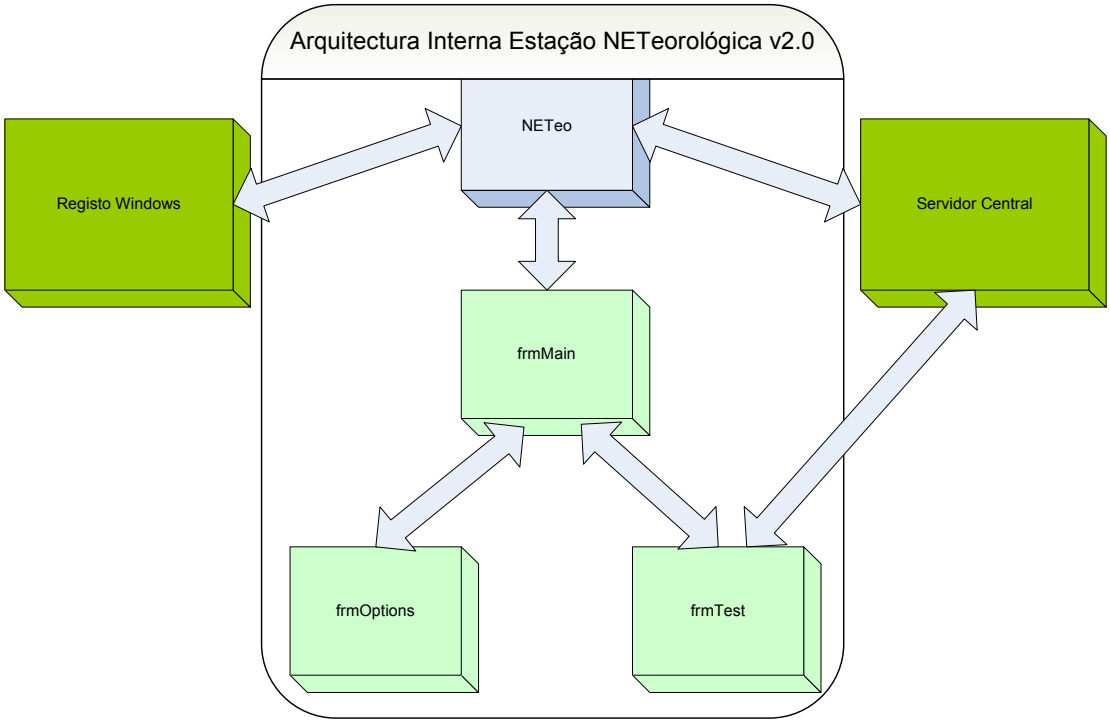


Figura 17 – Arquitectura interna da Segunda Versão da Apl. Residente (*StandAlone*)

Os diagramas reflectem as principais classes utilizadas, recursos externos e fluxos de dados entre estes.

Como é óbvio, em cada uma destas classes recorreram-se a várias classes e métodos da arquitectura .NET, tais como para a definição do ícone de ambiente de trabalho ou de temporizadores para a iniciação de teste.

Foram também definidas algumas estruturas para identificar o tipo de contrato e o progresso de um teste.

Note-se que as comunicações em geral são realizadas pela classe principal da aplicação (NETeo), sendo que o formulário de testes apenas comunica com o servidor central no contexto da realização do teste, transferindo ficheiros. Uma vez que este método será substituído pela implementação de *plug-ins* de medição, nem este formulário terá qualquer contacto com o servidor, directamente.

7.3.3. Descrição de Classes/Módulos

Existiu uma preocupação constante em organizar o código de uma forma tão estruturada quanto possível. Tal foi bastante claro na forma como foi organizado o código nas classes PHP do sítio *web*. No caso da aplicação residente, talvez não se possa dizer que esta organização seja tão evidente, visto possuir uma complexidade lógica inferior e o à-vontade com a

linguagem usada não é tão elevado, mas este cuidado perante a estrutura e organização do código está igualmente presente.

Passa-se a descrever as Classes e as suas funcionalidades mais relevantes.

NETeo:

Esta é a classe principal, que representa a aplicação.

Inicia o formulário principal, sem o mostrar, para que seja criado o ícone associado na bandeja de sistema.

Possui um método de comunicação com o servidor central (getServerString), num formato de pergunta/resposta em *String*.

São ainda implementados atributos virtuais, denominados em .NET como *Properties*, que permitem ler e escrever as configurações da aplicação (*Server, Username, Password, Periodicity, Autorun, Contract* e *ContracID*).

Esta classe define ainda alguns valores constantes tais como o endereço do servidor central, para ser usado nas medições.

frmMain:

Esta classe representa o formulário principal.

Nele é instanciado o ícone de bandeja de sistema, são controlados os temporizadores e permite-se o acesso aos restantes formulários ou à página *web* do projecto.

De notar que o formulário de testes é automaticamente instanciado no frmMain, para que apenas possa ocorrer um teste em paralelo.

A lista de contratos disponível é carregada com recurso ao método getServerString da classe NETeo.

frmTest:

Esta classe implementa a interface e a lógica de teste.

Este formulário é instanciado com uma referência para o formulário principal, para que possa deste ser consultado o valor do contrato seleccionado.

Realiza os testes efectuando as transferências através de comunicação directa com o servidor central.

frmOptions:

Aqui são disponibilizadas as opções da aplicação. Estas opções são guardadas no registo do Windows através das propriedades da classe principal.

Recorre-se também à classe principal para um pequeno teste de validade da autenticação do utilizador.

NETeoContract:

Esta classe permite definir o objecto que deve ser utilizado na caixa de escolha múltipla de contrato, registando o identificador do contrato e o nome, imprimindo o seu nome para visualização no controlo referido.

NETeoProgress:

Para permitir o uso de uma biblioteca de gestão de processamento assíncrono, denominada BackgroundWorker, revelou-se útil definir uma classe que represente o estado dum teste, para que este possa ser comunicado entre eventos assíncronos. Esta é a classe que cumpre essa tarefa.

7.3.4. Descrição das interfaces

Para o sucesso de qualquer aplicação Windows, é bastante importante que o desenho das suas interfaces seja bastante claro. Foi possível realizar interfaces com uma facilidade de interacção elevada para o utilizador. Para uma ainda maior clarificação, passa-se a descrever as 3 interfaces que a aplicação possui:

Interface Principal

1 – Indica ao utilizador a conta que está a ser utilizada neste momento

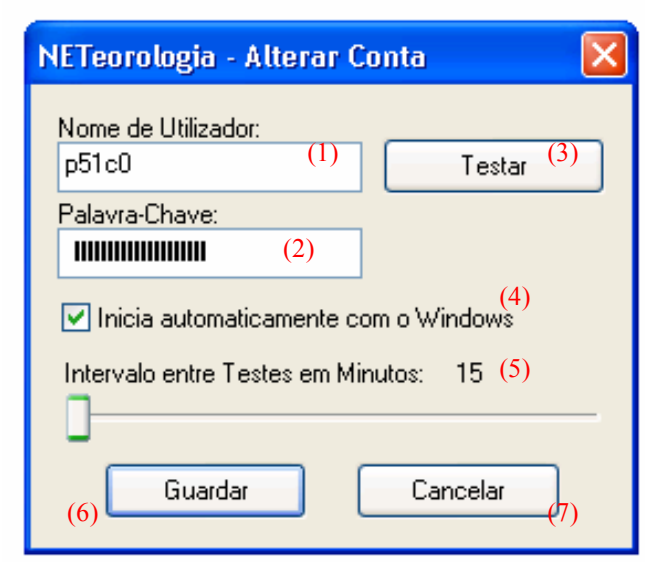
2 – Caixa de combinação que contém a lista de contratos associados ao utilizador, para que este indique, se necessário for, qual dos contratos está a ser utilizado

3 – Permite actualizar a lista de contratos a partir do servidor central

4 – Botão que permite aceder à interface de realização de teste explícito/esporádico.

5 – Botão que permite aceder à interface de configuração de conta/opções.

Interface de Opções



1 – Permite alterar o nome de utilizador a ser utilizado pela aplicação

2 – Permite configurar a palavra-chave a ser utilizada na autenticação perante o servidor

3 – Botão que permite ao utilizador testar os valores introduzidos em (1) e (2).

4 – Opção onde se indica se a aplicação deverá iniciar automaticamente com o Windows.

5 – Indicação e Barra de deslizamento para configuração do intervalo entre testes periódicos.

6 – Botão para fechar a interface, guardando as opções no registo

7 – Botão para fechar a interface descartando as alterações efectuadas

Interface de Teste

1 – Informação textual e em barra de progresso da fase principal em que se encontra o teste

2 – Informação textual e em barra de progresso da sub-fase em que se encontra o teste

3 – Resultados de *download*, *upload* e *ping* do teste.

4 – Botão que permite iniciar o teste.

5 – Botão que permite fechar a interface e regressar à interface principal.

7.3.5. Instalador

Qualquer aplicação minimamente complexa exige um processo de instalação simples e claro.

Com o auxílio do Visual Studio, foi criado, com facilidade, um projecto de instalador da Microsoft, segundo a tecnologia MSI. Este projecto detecta as dependências da aplicação e junta-as num ficheiro de instalação "NETeo.MSI". Para o caso de o utilizador não possuir um interpretador de ficheiros de instalação MSI é fornecido junto deste um pequeno ficheiro "Setup.exe" que executa o instalador.

A facilidade de execução desta parte do projecto foi uma surpresa, já que não existia qualquer experiência com este tipo de questões.

O instalador permite configurar o destino de instalação da aplicação e instala a maior parte das bibliotecas necessárias para a sua execução. A única biblioteca que não se encontra no instalador, a maior parte das pessoas já a possui, é a .NET Framework. De qualquer forma, o

instalador detecta se esta não está instalada e remete automaticamente o utilizador para uma página onde esta se pode obter gratuitamente.

Também detecta se já existe uma versão do produto instalada e, nesse caso, solicita que esta seja removida.

Após instalação, a aplicação é automaticamente executada, e, detectando que é a sua primeira execução, efectua algumas tarefas pós-instalação, tais como a definição de chaves no registo de sistema que permitirão guardar as configurações do programa e iniciá-lo automaticamente com o arranque do sistema operativo.

O arranque automático após instalação, aparentemente uma tarefa bastante comum e simples de resolver, não se revelou tal, tendo sido apenas implementada na segunda versão da aplicação.

Foi necessário implementar uma extensão da classe base .NET Installer que representa acções realizadas em período de instalação, na qual se analisam os parâmetros de instalação e se executa a aplicação de acordo com estes.

7.4. Modelo de Dados Persistente

Nesta secção, é apresentado o modelo de base de dados que serve de suporte a todo o funcionamento do sítio *web*, bem como da aplicação residente. Para completar a identificação das relações que deviam ser definidas na base de dados, as suas associações e os atributos que devem ter, e que permitem ao site disponibilizar as funcionalidades desejadas para os seus utilizadores, a base utilizada foi o mapa de conceitos previamente demonstrado.

O esquema seguinte mostra as tabelas, os seus atributos, identificadores (chaves primárias) e chaves estrangeiras, bem como as suas respectivas associações.

Para ter acesso ao código SQL de geração e inserção de valores por omissão, devem ser consultados o ficheiro *.sql fornecidos.

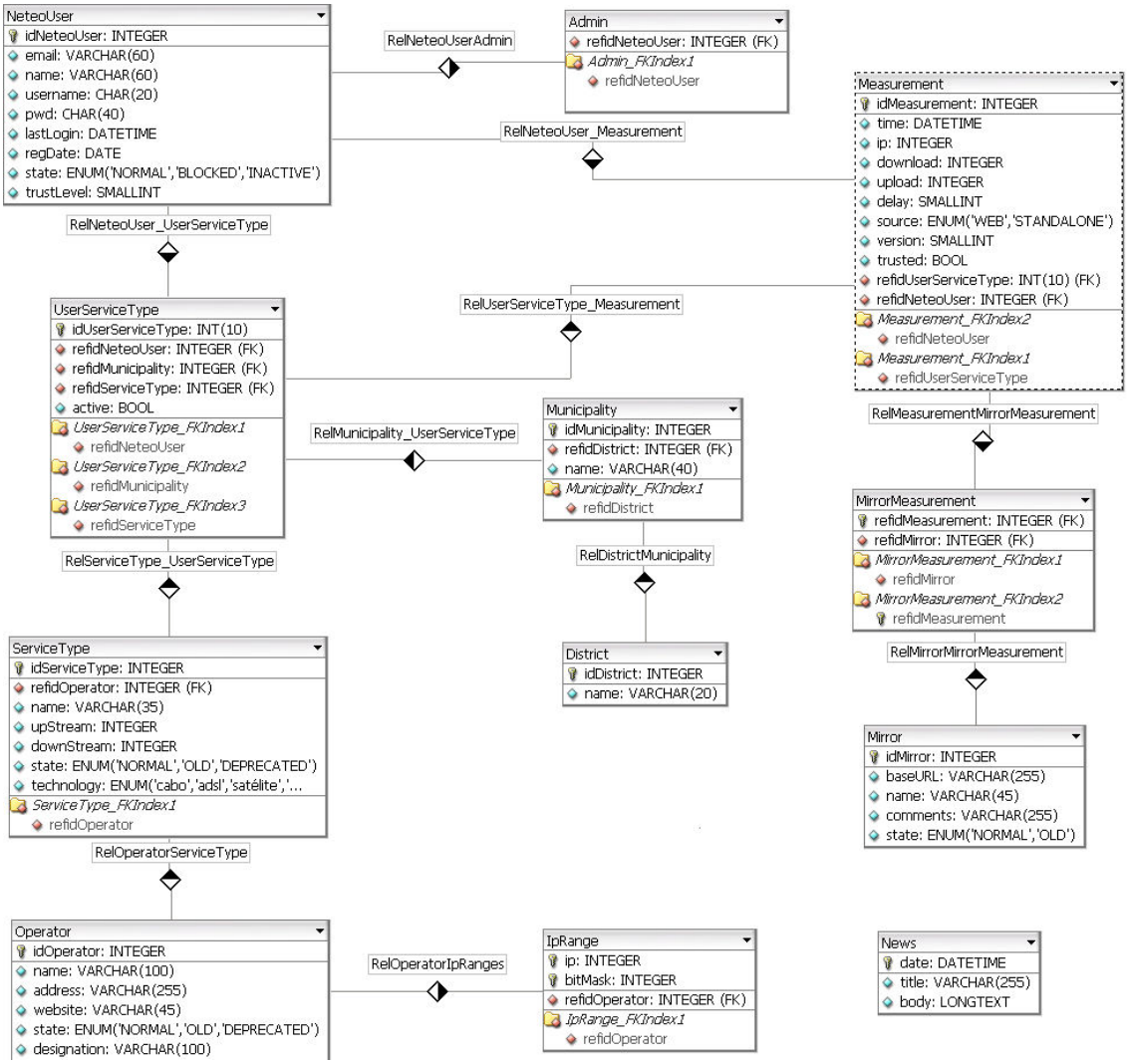


Figura 18 – Modelo de Dados

- **NeteoUser**: tabela usada para guardar dados relativos à conta de um utilizador que se registou no sistema através do site. Os campos que possui são os seguintes:
  - *email*: opcional (pode conter valor nulo), em que deverá ser inserido um endereço de correio electrónico, de tamanho variável, com um máximo de 60 caracteres.
  - *name*: campo opcional variável com um máximo de 60 caracteres para armazenar o nome (real) do utilizador.
  - *username*: campo de tamanho fixo com 20 caracteres, destinado ao nome de utilizador, que identifica de forma unívoca um determinado utilizador.
  - *pwd*: campo de tamanho fixo de 40 caracteres, para armazenar a palavra-chave de autenticação do utilizador. O valor armazenado resulta da aplicação de uma função MD5 ao texto inserido pelo utilizador.

- *lastLogin*: marca temporal da última acção feita pelo utilizador autenticado.
- *regDate*: marca temporal da sua data de registo no sistema. Por
- *state*: indicação do estado da conta do utilizador. Pode ser 'NORMAL', o que permite ter todos os privilégios associados a um utilizador registado, 'BLOCKED', que corresponde a não poder usar a sua conta (não é permitido o login), ou 'INACTIVE', que sinaliza uma conta não utilizada durante um dado intervalo de tempo de referência.
- *trustLevel*: campo que indica o grau de confiança que este utilizador apresenta, numa escala de 0 (confiança mínima) a 10 (confiança máxima).

- **Admin**: tabela para guardar referências relativas aos utilizadores que são administradores. Possui um campo apenas, com chave estrangeira para o identificador da tabela NeteoUser.

- **District**: tabela que armazena os nomes dos diversos distritos de Portugal. Além do identificador, possui apenas um campo de tamanho variável, com 20 caracteres, suficiente para guardar o nome de qualquer distrito do País (regiões autónomas identificados por R.A).

- **Municipality**: tabela onde são registados os nomes dos concelhos existentes em Portugal. Possui um campo de identificação, um campo *name* variável de 40 caracteres máximo para guardar o nome do concelho, e um campo que serve de chave estrangeira para o identificador do distrito em que se encontra esse concelho.

- **Operator**: tabela onde são guardados os dados relativos a um determinado operador. Possui os seguintes campos:
  - *name*: campo destinado ao nome com que o operador é conhecido comercialmente, sendo de tamanho variável com um máximo de 100 caracteres.
  - *address*: campo opcional em que é armazenado o endereço da sede social do operador. Trata-se de um campo de tamanho variável, para um máximo de 255 caracteres.
  - *website*: campo opcional para o endereço *Web* da página oficial do operador. Deverá ser começado por "http://". É de tamanho variável, com um máximo de 45 caracteres.

- *state*: campo enumerado, destinado a indicar o estado em que se encontra o operador. Pode ser: 'NORMAL', o que corresponde a um operador em actividade normal; 'DEPRECATED' para operadores que já deixaram de existir; 'OLD' para situações intermédias, de operadores "antigos", que mudaram de nome ou foram adquiridos por outros, podendo ainda existir utilizadores dele.
- *designation*: campo para guardar a designação oficial do operador, permitindo conhecer a entidade empresarial de que faz parte. O campo é de tamanho variável, com um máximo de 100 caracteres.

- **IpRange**: tabela destinada a guardar dados relativos a gamas de endereços IP. Para tal possui um campo *ip*, que guarda o valor do endereço de sub rede IP sob a forma de um inteiro (32 bits), um campo *bitMask*, para o valor da máscara de bits a aplicar, e um campo para que corresponde ao identificador do operador a que pertence esta gama de endereços IP, que serve de chave estrangeira.

- **ServiceType:** tabela destinada a conter os dados relativos aos tipos de serviço disponibilizados por cada operador. Tem os seguintes campos:
  - *name:* campo para guardar o nome pelo qual o serviço é conhecido, conforme o que se encontra no site oficial do operador a que pertence. Trata-se de um campo de tamanho variável, com tamanho máximo de 35 caracteres.
  - *upstream:* inteiro que indica o valor da largura de banda disponibilizada no sentido ascendente, ou seja, cliente-ISP. O valor deve ser inserido em *kilobits* por segundo (kbps).
  - *downstream:* inteiro correspondente à largura de banda no sentido descendente (ISP-cliente). A unidade do valor é o kbps.
  - *technology:* campo enumerado, indicando o tipo de tecnologia de acesso da oferta de serviço. Pode ser 'adsl', 'cabo', 'rdis', 'satélite', '*wifi*'.
  - *state:* 'NORMAL', referente a uma oferta em vigor, 'OLD' para tipos de serviço "*legacy*", que, não sendo parte da oferta comercial actual do operador, ainda são usados por utilizadores com contratos antigos; 'DEPRECATED' para tipos de serviço que já não existem de todo, não podendo ser escolhidos pelos utilizadores.

- **UserServiceType:** tabela que permite associar a um utilizador, um tipo de serviço e sua respectiva localização, ou seja, um determinado contrato de um utilizador.

Possui três chaves estrangeiras para essas mesmas tabelas, e ainda um campo booleano para indicar se o contrato está activo ou inactivo.

- **Measurement:** tabela que permite guardar os dados de uma medição, bem como a referência ao contrato com que foi feito. Possui uma chave estrangeira de referência ao contrato (UserServiceType) que foi, eventualmente, associado à medição. Os campos que a constituem são os seguintes:
  - *time:* marca temporal do dia e hora em que foi feita a medição.
  - *ip:* endereço IP com que foi feita a medição, que é guardado como inteiro (endereços IP têm 32 bits).
  - *download:* campo o valor da largura de banda de *download*, em *kilobits* por segundo (kbps).
  - *upload:* valor da largura de banda em *upload*, em kbps.
  - *delay:* valor do atraso RTT entre cliente e servidor de medição, em milissegundos
  - *source:* campo enumerado, relativo à fonte da medição – *WEB* para *applet* e STANDALONE para aplicação residente.
  - *version:* campo para guardar a versão da ferramenta de medição que foi usada.
  - *trusted:* sinaliza se a medição é confiável ou não.

- **Mirror:** tabela relativa aos dados referentes aos diversos servidores (*mirrors*) onde o *applet* se encontra disponível para realização de testes de largura de banda. Os campos que a compõem são:
  - *baseURL*: campo de tamanho variável (máximo de 255 caracteres) destinado a guardar o endereço onde se encontra alojado o *applet*, no servidor.
  - *name:* campo de tamanho variável, com um máximo de 45 caracteres, contendo o nome pelo qual é identificado o servidor.
  - *comments:* permite associar um breve comentário ao servidor, nomeadamente para informação dos administradores do sistema. Trata-se de um campo opcional, de tamanho variável, com 255 caracteres no máximo.
  - *state:* campo enumerado, para identificar o estado do servidor – NORMAL caso esteja activo ou OLD caso já não seja usado.

- **News:** tabela que guarda registos referentes às notícias. Possui os seguintes campos:
  - *date*: chave primária, corresponde à data e hora em que foi registada a notícia no sistema.
  - *title:* campo variável, de tamanho máximo 45 caracteres, relativo ao título da notícia.
  - *body:* campo de tipo *longtext*, para guardar o corpo da notícia.

7.5. Infra-estrutura de Auto-Updates

7.5.1. Abordagem

Optou-se por implementar a arquitectura separadamente da aplicação principal, recorrendo a uma pequena aplicação sem utilidade, mantendo as classes referentes aos *updates* perfeitamente isoladas.

Foi definida em primeiro lugar, a arquitectura básica que é relativamente simples, constituída por 3 componentes, o controlador das actualizações global, o formato de ficheiro XML no qual são transferidas e/ou guardadas informações sobre a necessidade de realizar actualizações, e o objecto que permite interagir dinamicamente com este tipo de ficheiros XML.

De referir ainda que se optou pela transferência simples de ficheiros em formato de biblioteca, para permitir a sua autenticação, sendo que os ficheiros podem reflectir tanto actualizações totais como parciais.

Após os métodos do controlador aparentarem um funcionamento correcto, foi criado um projecto paralelo que contém a estrutura base de uma actualização, devidamente autenticada.

Adaptou-se nesta altura o controlador de actualizações para que verifique a autenticidade da actualização, antes da sua aplicação.

Após a realização de testes satisfatórios, procedeu-se à adaptação da aplicação para a utilização desta arquitectura.

7.5.2. Descrição Técnica

Para a implementação desta arquitectura recorreu-se a duas classes.

Em primeiro lugar foi definida a classe **UpdateState**, que representa, tal como o nome indica, o estado de uma actualização.

Internamente, esta classe guarda os seus dados em formato XML, pelo que foi definido um serializador para um formato bem definido deste tipo de ficheiros. Existirão, do ponto de vista

da arquitectura, dois ficheiros deste tipo com relevância: o que é guardado localmente e que contém dados temporários sobre actualizações em curso, e o que é descarregado do servidor, que identifica a necessidade de realização de actualizações.

Este ficheiro, e o objecto associado, possuem os seguintes dados:

- CurrentVersion – Representa a versão mais recente disponível para transferência.
- Status – Representa o estado actual do processo de actualização:
    - Idle – Não foi ainda iniciado o processo;
    - Checking – A necessidade de actualização está a ser avaliada;
    - ErrorChecking – Ocorreu um erro durante a avaliação da necessidade de actualizar a aplicação;
    - Unavailable – Não existe nenhuma actualização para realizar;
    - Available – Está disponível, opcionalmente, uma actualização para transferência;
    - Required – Existe uma transferência disponível, e a esta é considerada essencial, não devendo a aplicação prosseguir a sua execução antes que esta actualização seja realizada;
    - Downloading – A actualização encontra-se a ser transferida;
    - ErrorDownloading – Ocorreu um erro durante a transferência da actualização;
    - Downloaded – A actualização foi transferida e pode ser aplicada assim que se deseje;
    - Applying – A actualização encontra-se a ser aplicada à aplicação;
    - ErrorApplying – Ocorreu um erro durante o processo de aplicação do *update*.
- DownloadPath – Endereço a partir do qual a actualização deverá ser descarregada.
- UpdateName – Nome interno da biblioteca que representa a actualização. A utilização do nome exacto com que a biblioteca foi publicada é essencial para a correcta verificação da sua autenticidade.
- ManifestPath – Endereço no qual pode ser encontrado o ficheiro que representa a necessidade e localização de nova actualização.

A outra classe implementada representa o cérebro do controlo de uma actualização, sendo uma classe global (*static*), já que apenas poderá, no contexto de uma aplicação, ocorrer um processo de actualização.

Esta classe, denominada **UpdateController**, cria o ficheiro de controlo interno, caso ele não exista, e expõe métodos e eventos que possibilitem à aplicação verificar a necessidade de actualização, seu estado, solicitarem a sua transferência e solicitarem a sua aplicação.

Existem duas maneiras fundamentais de uma aplicação utilizar esta classe para controlar os seus *updates*: sincronamente ou assincronamente.

Isto verifica-se na interface do primeiro método que deverá ser invocado pela aplicação:

```
static public UpdateStatus CheckUpdate(bool async)
```

A diferença prende-se com a instanciação do processo de verificação do *update* num fio de execução separado, para que a aplicação principal não tenha que aguardar pela sua conclusão. Esta diferenciação relaciona-se com a necessidade de algumas aplicações de verificarem a necessidade de actualização ao arrancar (não iniciando a aplicação enquanto não tiverem a certeza que não é necessária uma actualização – verificação síncrona) ou a necessidade de outras aplicações de realizarem esta verificação de forma periódica ou genericamente de forma não bloqueante (verificação assíncrona).

Para além de invocar este método, a aplicação deve registar-se em alguns eventos expostos pelo controlador, que permitem que a aplicação seja avisada do decorrer da actualização, autorizando ou não a sua continuação:

- StatusChecked – Este evento é disparado após a verificação da necessidade de actualização. Deve ser recebido pela aplicação para que seja informada do resultado desta verificação;
- UpdateAvailable – Ocorre sempre que exista uma actualização disponível, quer seja ela obrigatória ou não;
- UpdateRequired – Este evento é processado sempre que exista uma actualização obrigatória a realizar;
- Error – Ocorre sempre que tenha existido algum erro em qualquer fase do processo de verificação, transferência e aplicação de uma actualização;
- UpdateDownloaded – Indica que a actualização foi transferida e que se aguarda ordem para a sua aplicação;
- UpdateApplyed – Indica que a actualização já foi aplicada, permitindo à aplicação realizar algum tipo de limpeza da versão anterior.

Em função destes estímulos, a aplicação deverá decidir se e quando chamar os seguintes métodos:

- DownloadUpdate – Transfere a actualização disponível;
- ApplyUpdate – Aplica a actualização, extraindo os ficheiros desta e preparando a sua reinicialização;
- ExecuteAssembly – Arranca a nova versão da aplicação.

O método ApplyUpdate, possui uma complexidade elevada, pelo que se justifica detalhar o que este código realiza.

Em primeiro lugar, é verificado que a aplicação e a actualização foram assinadas com a mesma chave privada, recorrendo-se à chave pública da aplicação. Em simultâneo com esta verificação, a biblioteca é carregada em memória, sendo o seu conteúdo instantaneamente exposto ao controlador.

De seguida, o controlador verifica se existe algum método nesta biblioteca que possua os atributos *public* e *static*. Caso exista, interpreta-se este método como sendo uma rotina especial de extracção dos ficheiros e de actualização, sendo esta executada.

Não tendo ocorrido uma extracção personalizada, o controlador procede à extracção dos ficheiros que constam na actualização. Mas esta extracção não é pacífica. A maior parte das vezes que se realiza uma actualização pretende-se substituir ficheiros que estão neste momento bloqueados já que estão a ser utilizados pela aplicação, que ainda se encontra a correr. Existem métodos complexos para a resolução deste problema tais como a separação do controlador de *updates* numa aplicação completamente distinta, para que possa aplicar o *update* com a aplicação principal desactivada. Mas a solução, após descoberta, acabou por ser relativamente simples. Os ficheiros que já existam, durante a extracção, são renomeados (o que é permitido com os ficheiros abertos) para nomes temporários, com uma extensão predefinida.

Após a extracção, o próprio ficheiro de *update* é marcado para limpeza, sendo renomeado para a mesma extensão.

No que diz respeito ao arranque da nova versão da aplicação, sem recorrer a outro processo, e quando a versão anterior ainda não terminou (está a auto-actualizar-se), também não é um problema de simples solução.

Mesmo utilizando-se a renomeação de ficheiros, é sempre possível que o utilizador crie uma nova instância da aplicação enquanto se está a aplicar a actualização, o que poderia causar que se usassem em parte ficheiros antigos e em parte novos. Uma forma de resolução do problema é usando a capacidade de *Shadow Copy* dos domínios aplicacionais na *Common Language Runtime* (CLR).

A CLR corre o código gerido num recipiente lógico intitulado de domínio aplicacional (AppDomain) [odetocode2004]. É possível que vários AppDomains sejam instanciados num mesmo processo, permitindo a execução de duas aplicações independentes em termos lógicos, num mesmo processo. Quando se cria um AppDomain é possível fazer *Shadow Copy* do mesmo, o que significa que quando esta nova aplicação carrega os seus ficheiros, estes são copiados e carregados a partir de uma pasta escondida temporária.

É através desta técnica que a plataforma ASP.NET [odetocode2004] permite actualizar o código enquanto este está a ser executado por vários utilizadores.

Os domínios aplicacionais são também bastante úteis para a re-execução da aplicação.

É ainda preparado o contexto da aplicação, tal como os argumentos de entrada, para posterior execução no método ExecuteAssembly().

Do ponto de vista da aplicação, registam-se os controladores de eventos que darão resposta às diferentes fases de controlo de actualização, bastando, neste caso, limitar o recurso aos testes durante estas fases, e dar origem, sequencialmente, à próxima fase da actualização.

Após a actualização aplicada, a aplicação termina, para dar origem à nova versão.

7.5.3. Tutorial de Distribuição

Para estruturar e exemplificar a criação de uma actualização, foi criado um projecto de criação de uma actualização.

Este projecto contém 2 ficheiros, para além daqueles que se pretenda incluir na actualização:

- StrongName.cs – Para que a biblioteca seja criada, utilizando-se *StrongNaming*, é necessário que exista pelo menos uma classe, ainda que esta não implemente absolutamente nada. É possível definir nesta classe, ou noutra classe qualquer que se deseje criar, um método que defina uma extracção personalizada dos ficheiros, bastando adicionar um método *public* e *static*.
- NeteoKey.pfx – Para facilitar a utilização da mesma chave nos dois projectos (aplicação principal e actualização), recorreu-se à criação de um ficheiro que contém a chave privada.

Para criar uma actualização basta anexar todos os ficheiros (os executáveis já compilados ou outros recursos) indicando em cada um dos ficheiros que estes devem ser comprimidos na biblioteca (opção *BuildAction*="*Embedded Resource*").

Compilado o ficheiro DLL, basta editar o ficheiro XML para que reflicta a nova versão, indicando a nova necessidade e localização onde este ficheiro deve ser descarregado.

7.6. Infra-estrutura de Plug-ins

7.6.1. Abordagem

Em primeiro lugar, foram realizados vários testes periciais no uso de *plug-ins*.

Numa dada aplicação, define-se a interface que o *plug-in* deverá ter, identificando os métodos, atributos e propriedades que a aplicação principal pode invocar bem como a interface da aplicação perante o *plug-in*, no qual se definem os pontos de contacto entre as duas entidades computacionais, que são da iniciativa do *plug-in*.

É ainda de utilidade recorrente a utilização de uma interface gráfica de escolha de *plug-in*, no qual são listadas as bibliotecas disponíveis num determinado local que implementam a interface definida.

Após ter sido gerada uma pequena aplicação de teste que cumpria estes requisitos, e estando consolidados os mecanismos que permitem a implementação de extensibilidade, tentou-se verificar as possibilidades de abstracção de uma arquitectura deste tipo para que possa ser facilmente adaptada noutras situações e noutras aplicações.

Mas esta tarefa revelou-se de uma complexidade elevadíssima. A definição das interfaces de comunicação entre entidades é sempre muito específica da aplicação em concreto que se estiver a programar, pelo que se torna virtualmente impossível a definição de métodos com abstracção suficiente para que se possam aplicar em várias aplicações. No entanto, o tipo de tarefas a realizar no âmbito da identificação, teste e instanciação de um *plug-in,* deverão ser relativamente semelhantes de aplicação para aplicação, pelo que existem algumas partes do trabalho que foram separadas numa biblioteca de gestão de *plug-ins,* contendo funções de listagem e de carregamento dinâmico de objectos que implementam uma determinada interface.

Procedeu-se então à identificação dos métodos e propriedades que iriam dar origem às duas interfaces de programação anteriormente referidas. Esta foi a tarefa crucial neste trecho de implementação, já que após definidas as interfaces e criados os primeiros *plug-ins*, estes não são facilmente alteráveis sem que se causem quebras de compatibilidade.

Foi realizado um esforço de generalização e abstracção, para permitir uma maior impermeabilidade a futuros problemas de incompatibilidade. (Quanto mais abstractos forem os métodos, mais facilmente poderão ser usados para funcionalidades que se venham a achar necessárias).

Definidas as interfaces, foi implementado um *plug-in* de teste, que ainda que não realize qualquer medição real, permite comprovar o correcto funcionamento da arquitectura implementada.

Tendo sido bem sucedida a implementação, as atenções voltaram-se para uma outra questão, associada à segurança. Era necessário garantir a distinção entre plug-ins certificados e plug-ins por certificar, de forma a que os dados produzidos por um ou outro caso possam ser distinguidos nos processos de análise da informação gerada pelos utilizadores globalmente.

O controlador de *plug-ins* passou a analisar igualmente a autenticidade do código de uma aplicação perante uma chave pública indicada pela aplicação.

7.6.2. Descrição Técnica

Tecnicamente, a implementação da extensibilidade concretizou-se na definição de duas interfaces e na implementação de uma classe gestora de plug-ins.

A interface que um *plug-in* deverá implementar (INETeoMeasurementPlugin) possui os seguintes métodos:

- void doTest() – Este método deverá realizar os testes de medição, guardando o progresso e os resultados ao longo dos testes, numa classe própria para o efeito, a NETeoProgress. Este método é iniciado assincronamente, pelo que o *plug-in* deverá reportar o progresso, sempre que este sofra alguma alteração, através do respectivo método da interface da aplicação perante o *plug-in*.
- String getName() – Este método fornece um nome para o *plug-in*, que poderá ser apresentado ao utilizador.
- void Initialize(INETeoFrmTest application) – Este deverá ser o método de inicialização do *plug-in,* sendo-lhe fornecida uma referência para o formulário com o qual poderá interagir e ao qual deverá reportar o seu progresso.

Note-se que o nível de abstracção desta interface, sendo bastante elevado, permite uma mais fácil eventual evolução das técnicas envolvidas. Por exemplo, para que, no futuro, se realiza-se a medição de um outro tipo de dados adicional, tal como a variação do atraso, bastaria estender a classe NETeoProgress para incluir esse valor, não se quebrando qualquer compatibilidade com *plug-ins* desenvolvidos anteriormente.

A interface que o *plug-in* poderá usar, para fornecer e obter dados do diálogo de realização de teste (INETeoFrmTest) possui os seguintes métodos:

- reportProgress(NETeoProgress progress) – É através deste método que o *plug-in* deve reportar o estado da evolução do teste, quer para assinalar que este já terminou, quer para fornecer dados durante o teste que permitam ao formulário fornecer uma indicação visual do mesmo.
- String getPath() – Permite ao *plug-in* obter o endereço base onde se encontra o servidor central implementado.
- String getDownloadFile() – Permite ao *plug-in* obter o endereço de um ficheiro disponibilizado no servidor central, para teste de velocidade de transferência.

Existem ainda algumas tarefas associadas à gestão de *plug-ins*, que resultaram na implementação de uma classe denominada PluginController.

Esta classe implementa os seguintes métodos públicos:

- Plugin[] listPlugins(String strPath, String strInterface, strKey publicKey) – Dada uma determinada directoria, este método analisa os ficheiros .dll existentes na mesma, analisando os mesmos no sentido de identificar aqueles que implementam a interface identificada no segundo argumento. São ainda distinguidos os *plug-ins* que se encontram autenticados com a chave fornecida dos que não se encontram autenticados. Este método é composto por duas etapas, a de listagem de ficheiros, e o teste individual a cada ficheiro. Esta é a questão mais complexa, pelo que foi separada para um método privado com a seguinte assinatura: Plugin TestPlugin(String strDll, String strInterface, String strKey). Neste método, em primeiro lugar verifica-se se o dll se encontra certificado, verificando se este se encontra assinado com uma determinada chave, através da tecnologia de *Strong Naming*. Analisam-se de seguida os vários tipos definidos na biblioteca importada, verificando a existência de alguma classe que implemente a Interface fornecida. Caso exista, é criada uma instância de uma estrutura Plugin que contém a localização do ficheiro, a classe que implementa a Interface e a indicação da existência ou não de creditação do ficheiro.
- Object loadPlugin(Plugin p) – Este método carrega um determinado *plug-in*, retornando a referência para uma instância da classe respectiva.

No que diz respeito à execução deste controlador, na aplicação, adicionou-se um formulário de listagem de *plug-ins*, na qual se poderá escolher o elemento a utilizar, sendo disponibilizada sempre uma versão interna por omissão.

Foi ainda adaptado o código para que o *plug-in* seleccionado seja identificado em conjunto com as restantes opções, no registo do sistema.

Sempre que é necessário iniciar um teste, solicita-se ao controlador de *plug-ins* a instanciação de uma classe que é executada de forma muito similar à que já se fazia anteriormente.

7.6.3. Tutorial de Distribuição

O conceito de interface torna a implementação de um *plug-in* bastante simples. Basta que seja criado, em qualquer linguagem .NET, uma biblioteca com uma classe que implemente a interface INETeoMeasurementPlugin.

Para exemplificar, foi criado um projecto em C#, com a estrutura já definida e comentada, sendo apenas necessário implementar o código que se pretenda.

De notar que para certificar uma biblioteca, basta adicionar a um projecto a assinatura com a chave privada NETeo distribuída, a mesma utilizada nos variados projectos e também já incluída no projecto *template* de *plug-in*.

O *template* possui assim os seguintes ficheiros:

Este projecto contém 2 ficheiros, para além daqueles que se pretenda incluir na actualização:

- MyNETeoPlugin.cs – Neste ficheiro encontra-se estruturada uma classe com o mesmo nome, que implementa a interface INETeoMeasurementPlugin.
- NeteoKey.pfx – Para facilitar a utilização da mesma chave nos vários projectos, recorreu-se à criação de um ficheiro que contém a chave privada.

8. Considerações sobre técnicas de medição

O principal objectivo do projecto consistia em desenvolver uma plataforma para medição da qualidade de serviço da banda larga, na perspectiva do utilizador. A qualidade é aferida a partir da avaliação da capacidade de transferência de dados, quer em termos de transmissão (sentido cliente-servidor) quer em termos de recepção (sentido servidor-cliente), e da aferição do tempo de atraso RTT entre o cliente e o servidor.

É feita uma separação entre o teste de transmissão e de recepção, visto que em ambos os casos é gerado algum tráfego no sentido inverso, se bem que em quantidade muito reduzida. Desta forma, quando é feita uma transferência de dados num sentido, são enviados *acknowledges* TCP (ACKs) no sentido inverso, que servem para sinalizar a recepção correcta dos diversos segmentos de dados, devendo o seu atraso ser o menor possível em relação à recepção dos dados. Caso haja muito tráfego no mesmo sentido dos ACKs, estes correm o risco de ser atrasados. O emissor dos dados pode concluir então que está a enviar dados a uma taxa acima do permitido pelo receptor, levando-o a reduzir a sua taxa de transmissão. Este problema tem particular incidência em ligações assimétricas, como é o caso do cabo ou ADSL.

Não executando os testes de *download* e *upload* em simultâneo procura-se eliminar este problema.

Em relação à aferição do atraso, é utilizada a globalmente conhecida ferramenta de rede *ping*, que permite obter o atraso de ida-e-volta (*Round-trip-time*), bem como a taxa de perda de pacotes entre dois nós de rede. Faz uso de pacotes ICMP "*echo request"* e "*echo reply"*. A versão Linux, usada no nosso sistema, devolve o valor mínimo, médio, máximo e de desvio dos atrasos medidos.

Convém referir que existem certos ISP's que filtram os pacotes do primeiro tipo, por razões de segurança, o que inibe o uso desta ferramenta. As configurações do *router* e/ou *firewall* no lado do utilizador também podem estar definidas para filtrar este tipo de pacotes.

No caso do *applet* desenvolvido, as técnicas de medição utilizadas consistem num mecanismo "simples" de envio/recepção de dados com um tamanho conhecido (através da transferência de ficheiros via HTTP), na medição do tempo dispendido para essa transferência e no cálculo da largura de banda a partir desses dois valores. Trata-se de um método que permite medir a quantidade de largura de banda disponível para o utilizador, devendo-se ter em conta que, no caso de o utilizador se encontrar a realizar outras transferências, a medição irá estar em concorrência com essas transferências, pelo que os resultados obtidos pelo teste poderão ser inferiores à efectiva largura de banda disponibilizada pelo operador ao cliente. A largura de

banda disponível pode então ser definida como o débito máximo ao nível da camada IP que um percurso entre dois nós de rede pode oferecer ao fluxo, tendo em conta o tráfego cruzado (*cross-traffic*) que esse percurso apresenta no momento.

Por outro lado, os testes efectuados para diversos tipos de serviço em diversas localidades, mostram que os resultados dos testes são muito próximos da capacidade contratada pelo cliente, existindo inclusive casos de contratos em que os valores medidos foram consistentemente superiores ao máximo esperado.

As figuras seguintes mostram as médias dos resultados das medições, em percentual em relação ao contratado para o *download* e *upload*, para diversos contratos (conjugações distintas de tipo de serviço/localização).

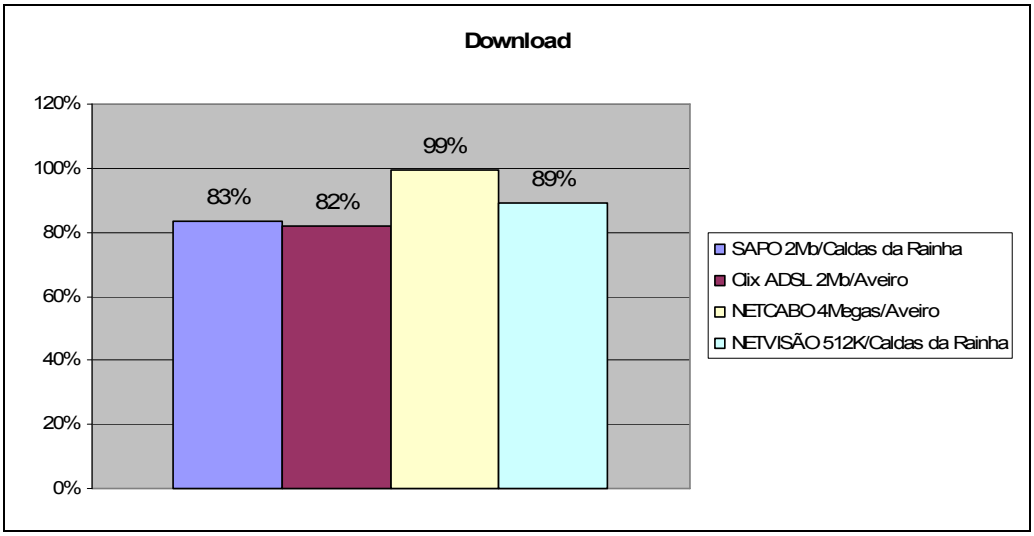


Figura 19 – Medições de *Download* Exemplificativas

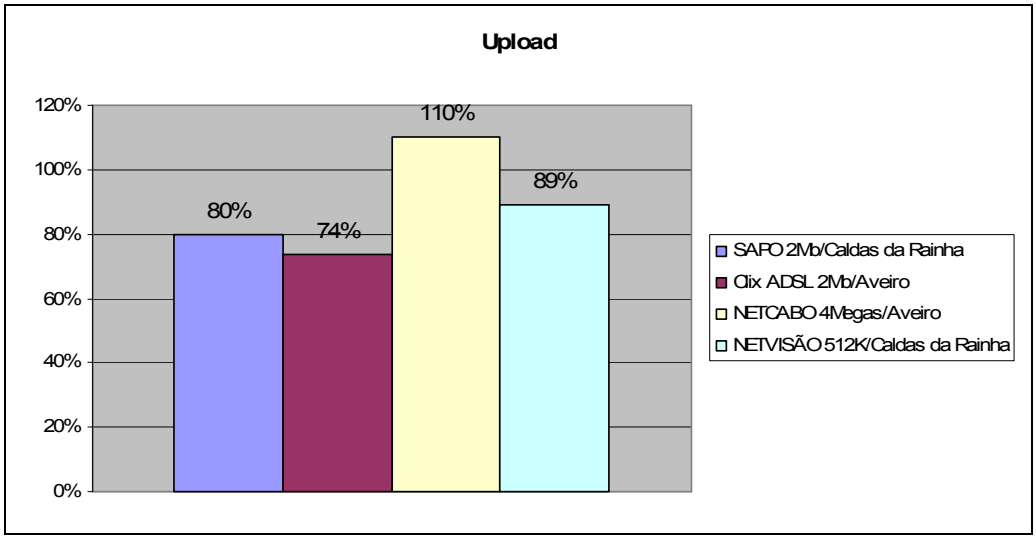


Figura 20 – Medições de *Upload* Exemplificativas

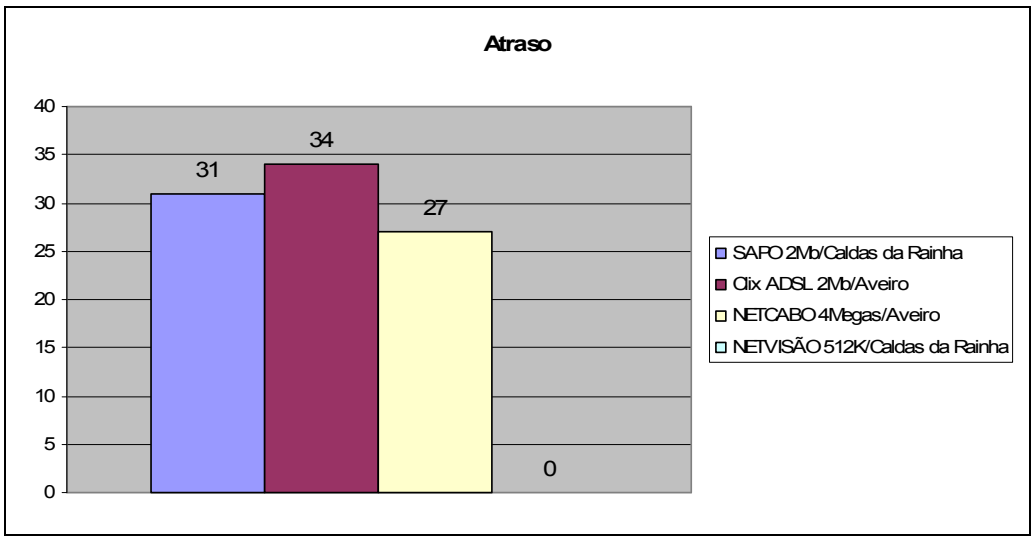


Figura 21 – Medições de *Round-Trip-Time* Exemplificativas

(nota: para o contrato Netvisão 512K/Caldas da Rainha, não foi possível obter o valor do atraso)

Estas médias reflectem a execução do teste em ambiente "real", ou seja, foram realizados a partir de ambientes domésticos heterogéneos, onde se incluem ligações directas ao equipamento do operador (modem cabo/ADSL ligado via porta *Ethernet* ou USB ao computador do utilizador) e ligações indirectas/intermediadas, usando mecanismos de NAT em topologias de rede diversificadas, onde se incluem o uso de equipamentos de nível 2 e 3 do modelo OSI (*hubs*, *switches* e *routers*). Desta forma, os testes aproximam-se significativamente dos ambientes e infra-estruturas tipicamente usados pelos utilizadores.

Não foi identificada nenhuma diferença entre o uso de ligação directa e intermediada em relação aos valores das medições. Isto deriva do facto da largura de banda oferecida pelas

infra-estruturas usadas (redes *Ethernet* 10/100) ser bastante superior ao débito da ligação de acesso à Internet, não criando assim um *bottleneck* no caminho dos dados.

Um problema decorrente do uso de ligações intermediadas prende-se com o facto de serem ligações normalmente partilhadas entre vários utilizadores, o que pode significar que durante o teste, outros utilizadores possam estar a utilizar a ligação Internet. O utilizador deverá estar alerta para este facto, para poder proceder a uma análise informada dos resultados obtidos na medição.

A técnica usada para aferição da qualidade da ligação em banda larga através da aplicação residente foi alvo de análise detalhada, devido à sua característica de medição periódica, em contraponto com a medição com um carácter mais esporádico da ferramenta *applet*.

Usando a aplicação residente, chegou-se à conclusão que o tráfego gerado entre o cliente e o servidor poderia ser significativo, tendo em conta a primeira abordagem que fizemos ao problema. Nesta abordagem, partimos de uma situação em que o utilizador teria o computador permanentemente ligado (o que não é raro actualmente) e a aplicação sempre a correr, efectuando então o teste a cada 30 minutos. Se para cada teste fosse efectuado o *download* de apenas 1 *MByte*, isso corresponderia a um total superior a 1,4 *GBytes* no final do mês, o que foi considerado excessivo, se atendermos aos limites de tráfego que os operadores impõem aos seus clientes, que nalguns casos são da ordem dos 4 GB.

Visto que esta abordagem era claramente insatisfatória, procuraram-se alternativas, que seguiram por duas vertentes: estudo de técnicas de medição alternativas, mais sofisticadas, para aferição da largura de banda, envolvendo uma menor geração de tráfego, e alteração das características da abordagem inicial (nomeadamente a periodicidade dos testes), usando a mesma técnica dessa abordagem.

Técnicas de Medição Alternativas

O estudo de técnicas de medição alternativas baseou-se principalmente no trabalho desenvolvido pelo professor Constantinos Dovrolis, professor do Georgia Tech, que possui ampla bibliografia sobre o tema de *"Bandwidth Estimation"*. As principais referências consultadas foram: "*What do packet dispersion techniques measure*?" [packets] e "*Bandwidth Estimation: metrics, measurement techniques, and tools*" [bwest]. Nestes trabalhos são apresentadas as métricas associadas à medição da largura de banda, técnicas para estimar a largura de banda de uma ligação através do uso de *dispersão de pacotes*, e ferramentas que implementam essas técnicas e metodologias. Foi considerado o uso destas técnicas pois, num plano teórico, ofereciam a possibilidade de obter uma estimativa aproximada do valor da largura de banda, criando um *overhead* de tráfego associado reduzido, menos intrusivo do que no caso da técnica usada inicialmente.

As métricas definidas em [bwest] são as seguintes: capacidade, correspondendo à largura de banda máxima disponibilizada pela ligação ou percurso, sendo então igual à capacidade da ligação com largura de banda mais estreita (*narrow link*); largura de banda disponível, que se refere à largura de banda não utilizada pela ligação, ou seja, a menor quantidade disponível ao longo do percurso (*tight link*); *"Bulk Transfer Capacity"*, relacionada com o débito máximo atingido por uma transferência TCP.

Em relação à capacidade (máxima) de uma ligação/percurso, é de notar que o seu valor tende a ser o mesmo ao longo de um intervalo de tempo considerável, sendo alterado quando ocorre o melhoramento da linha por exemplo.

Posto isto, é a capacidade ou largura de banda disponível que mais interessa, métrica que varia consideravelmente ao longo do tempo. O seu valor depende da quantidade de *cross-traffic* que exista na ligação ou percurso, indicando a largura de banda que pode ser usada efectivamente pelo cliente. Caso a utilização da ligação seja nula ou próxima desse valor, a largura de banda disponível tende para o valor da capacidade. É devido a isto que o utilizador do sistema é aconselhado a reduzir ao mínimo (preferivelmente deverá ser nula) a utilização da ligação por outras aplicações concorrenciais às ferramentas de medição, visto que nestas condições os resultados dos testes reflectem a largura de banda que o utilizador tem efectivamente disponível, partindo do princípio, que deve ser assegurado sempre, de que o servidor possui largura de banda no sentido ascendente, ou seja, de transmissão, suficiente para que seja possível atingir o débito máximo por parte do cliente, de modo a que seja a sua ligação o *tight link*, e não outra do percurso.

É esta capacidade disponível que indica a qualidade do serviço prestado ao cliente, devendo aproximar-se o mais possível do valor contratado para a oferta em causa. Desta forma justifica-se o percentual usado na informação estatística referente às medições dos utilizadores, que pode ser consultada através do mapa dinâmico das regiões ou dos gráficos diários/mensais.

Em [bwest] são apresentadas algumas metodologias e ferramentas para estimar as métricas da largura de banda, que foram testadas por nós para podermos analisar a sua eficácia.

Os testes destas ferramentas foram realizados em ambiente "semi-controlado" e em ambiente "real". No primeiro caso foi feita uma ligação VPN entre uma rede doméstica e a rede da UA, sobre uma ligação Clix ADSL de 2 *Mbits/s*, sendo o teste executado entre um computador da rede doméstica e o servidor Ares do IT, que estavam assim dentro da mesma rede privada virtual; no segundo caso a ligação entre a rede doméstica e o servidor Ares era directa, sem recurso a VPN, reflectindo assim o ambiente usual de um utilizador.

Para cada uma das ferramentas, apresenta-se em seguida uma breve descrição, bem como as nossas impressões, os resultados que obtivemos e o *overhead* de tráfego associado, que foi

dividido em três classes: baixo (inferior a 100KB), médio (entre 100 e 1000 KB) e alto (superior a 1000 KB).

De referir que em ambiente real, não foi possível a obtenção de qualquer estimativa para as métricas alvo.

Pathrate – estima o valor da capacidade máxima de uma ligação/percurso, através do uso *packet pairs/packet trains*. Deve ser instalado em ambos os extremos do percurso. A sua instalação decorreu se problemas significativos, de notar apenas a existência de *warnings* durante a compilação. Em ambiente controlado, os resultados estimaram com precisão a capacidade em 70% dos testes. Em ambiente real ocorreram erros, não tendo sido obtidos quaisquer resultados. *Overhead* médio.

Spruce – é uma ferramenta que permite estimar o valor da largura de banda disponível, sendo necessário o conhecimento prévio da capacidade da ligação. A sua instalação decorreu sem problemas no servidor Ares, apresentando erros no computador doméstico. Quer em ambiente controlado quer em ambiente real não foram obtidos valores conclusivos. *Overhead* baixo.

Pathload – usa uma metodologia de SLoPS, devolvendo uma gama de valores para os quais a estimativa é válida. Requer o acesso a ambos os nós extremos. A sua instalação foi feita sem problemas de maior. Os resultados em ambiente controlado estimaram a largura de banda disponível correctamente em cerca de 70% dos casos, não havendo resultados em ambiente real. *Overhead* alto.

IGI – metodologia idêntica à usado pelo Pathload, procurando estimar a largura de banda disponível entre extremos. Instalação correcta. Em ambiente controlado foi possível estimar os valores da largura de banda com êxito em 65% das vezes. *Overhead* alto.

VR – ferramenta desenvolvido por alunos de projecto do DET-UA, para testar a capacidade *hop* a *hop*. Não foi possível testá-la devido à ocorrência de erros durante a sua execução (*segmentation fault*).

(Esta ferramenta não constava na lista apresentada em [bwest], tendo sido fornecida directamente, cortesia da professora Susana Sargento).

Como se pode verificar, os resultados das estimativas não garantem a obtenção de valores satisfatórios, especialmente se se tiver em conta que em ambiente real, idêntico ao dos

utilizadores alvo do sistema, a execução das ferramentas não teve sucesso de todo, tendo sido impossível a obtenção de valores para análise.

Esta situação deriva, em parte pelo menos, das condições impostas pelas ferramentas, que foram desenhadas para uma utilização em ambiente laboratorial controlado, sendo particularmente susceptíveis a *cross traffic* com variações temporais bruscas. Em utilização normal, as condições pressupostas para a execução são muito difíceis ou impossíveis de verificar.

Outra questão relacionada com estas ferramentas prende-se com o facto de terem sido desenvolvidas em ambiente Linux, usando as bibliotecas específicas desse Sistema Operativo. Para a sua adaptação sistema NETeorologia, seria necessária a migração para o ambiente Windows, o que envolveria um esforço considerável.

Em relação à característica que mais interesse causava nestas técnicas, de gerar um tráfego reduzido para estimar os valores pretendidos, é de salientar o facto de nalgumas situações esse tráfego ser significativo, aproximando-se do gerado pela técnica inicial.

Tendo em conta todas estas questões, ponderou-se se seria compensador o esforço necessário para efectuar a adaptação das metodologias utilizadas pelas ferramentas testadas à nossa aplicação, considerando ainda que os resultados dos testes não foram satisfatórios, revelando que estas técnicas não oferecem garantias suficientes de obtenção de estimativas próximas dos valores reais da largura de banda. A conclusão a que se chegou foi a de não recorrer a estas metodologias e técnicas, visto que o risco envolvido em desenvolver uma solução que as adoptasse foi considerado elevado, para a compensação que se obteria.

Prosseguiu-se então com a abordagem da técnica inicial, procedendo-se a algumas alterações, a nível da funcionalidade oferecida pela aplicação, de modo a obviar o problema do tráfego gerado.

Essas alterações consistiam em oferecer ao utilizador a possibilidade de alterar a periodicidade dos testes conforme a sua vontade, ficando este com maior liberdade para definir a quantidade de tráfego criado pelas medições. Desta forma, mantém-se uma técnica mais simples mas que provou ter tido resultados fiáveis, como se pode verificar pelos nossos testes e pela sua adopção noutros projectos similares [anacom2005].

9. Conclusões e Futuras Evoluções

Neste capítulo serão apresentadas as elações retiradas durante a implementação das várias fases e componentes da plataforma, bem como indicadas as possíveis evoluções das mesmas.

Na primeira secção, será feita uma análise geral sobre a plataforma, com especial ênfase no servidor central. As outras secções, uma vez que o detalhe assim exige, descrevem as conclusões retiradas durante a implementação, respectivamente da Aplicação Residente, Infra-Estrutura de *Auto-Updates* e da Infra-Estrutura de Plug-ins.

9.1. Gerais

O estado actual de desenvolvimento do projecto permite apresentar aos utilizadores uma plataforma que cumpre no essencial os objectivos principais propostos no início. Foram desenvolvidas as ferramentas para analisar a qualidade do serviço de banda larga em tempo real, quer através de medições esporádicas a partir do *applet* Java quer através do uso da aplicação residente, que permite automatizar os momentos de execução dos testes de largura de banda, de modo a seguir as suas variações ao longo do tempo.

A plataforma disponibiliza também um *website* de suporte, onde pode ser consultada a informação relativa às medições, procedendo-se a um tratamento estatístico, com possibilidade de definição de diversos critérios de filtragem, o que vai de encontro às expectativas dos utilizadores. Foram igualmente criadas facilidades de gestão para as contas dos utilizadores registados bem como para a gestão de operadores/tipos de serviço, notícias, utilizadores e *mirrors*.

Em relação ao fórum de discussão, recorreu-se a uma solução *open-source***,** porventura das mais utilizadas a nível mundial [phpBB].

Em relação à passagem da plataforma para uma utilização pública alargada, em que se pode prever um acesso simultâneo de uma quantidade significativa de utilizadores, deve-se ter em conta qual a máquina a usar como servidor, assim como a largura de banda, em ambos os sentidos, que é oferecida. Actualmente é usado um servidor residente nas instalações do IT, no *campus* da UA, em ambiente partilhado, visto que o projecto ainda não passou para o público. Caso isso aconteça (e existe esperança de que tal venha a acontecer) é necessário arranjar uma infra-estrutura que consiga suportar a quantidade de tráfego prevista, quer em termos de capacidade de processamento e armazenamento de dados, quer em termos de capacidade de largura de banda.

O próprio conceito de *mirror* associado à plataforma procura tornar mais escalável todo o sistema, de modo a que sejam oferecidos aos utilizadores vários servidores a partir dos quais seja possível executar os testes. Seria interessante haver uma associação com uma entidade externa credível (por exemplo, o regulador ANACOM seria uma boa opção), com capacidade

suficiente para garantir medições fiáveis, com servidores alojados em locais onde fosse possível oferecer igualdade de circunstâncias para todos os operadores presentes no mercado, não privilegiando nenhum em específico. Considera-se que desta forma a plataforma ganharia maior visibilidade e credibilidade, bem como um suporte físico capaz de garantir medições correctas, promovendo assim o seu uso por parte dos utilizadores.

No que diz respeito ao *applet*, ele cumpre com os requisitos pretendidos, sendo apenas de notar que existe um pequeno problema com a seu interface gráfica, nomeadamente na parte em que mostra o progresso do *upload* de ficheiros. Devido a limitações da tecnologia utilizada, não é possível fazer o progresso da barra à medida que vão sendo enviados os dados. Os valores apresentados são aleatórios, servindo apenas para fornecer ao utilizador um *feedback* indicando que o teste continua a decorrer. Foi adoptada esta solução pois foi considerado mais negativo que a barra de progresso aparecesse sempre vazia, induzindo em erro o utilizador, levando-o a pensar que o *upload* de dados não estava a correr como esperado. Desta forma, o utilizador fica com a noção de que o teste está a decorrer normalmente, o que é verdade.

Outra questão relacionada com o *website* diz respeito às ferramentas de validação de registos, responsáveis pela verificação da correspondência correcta entre as medições feitas e os contratos associados. Essa validação é muito importante, visto que a plataforma só representa uma mais-valia para os utilizadores caso as medições reflictam a real Qualidade de Serviço da oferta disponibilizada pelos operadores. Deste modo, é fundamental que hajam mecanismos que permitam verificar se a associação entre medição e contrato de serviço é realista ou não.

As ferramentas actualmente desenvolvidas conseguem identificar o operador que está a fornecer o serviço ao utilizador que realiza o teste, através duma comparação entre o endereço IP do utilizador e as gamas de endereços registadas na base de dados. Através da identificação do operador, pode-se em seguida restringir os contratos listados para possível associação à medição de modo a que sejam apenas relativos a esse operador e que, além disso, tenham uma capacidade de recepção, ou seja, no sentido *downstream*, suficiente para atingir o valor de *download* medido. Esta capacidade de recepção não corresponde exactamente ao valor de *downstream* da oferta, mas sim a um valor até 50% superior a esse, dado que é normal que os limites indicados pelo prestador do serviço possam ser ultrapassados. Com estas restrições, limitam-se as opções do utilizador, potenciando assim uma escolha correcta.

Em termos gerais, a plataforma desenvolvida cumpre os requisitos definidos inicialmente e apresenta condições que ajudam na sua manutenção e em futuros desenvolvimentos, dado que além deste documento de referência, foram feitos esforços para dotar de comentários todo o código criado. Considera-se que estes são suficientes para se ter uma documentação clara e objectiva de todos os componentes de software elaborados.

9.2. Aplicação Residente

O cuidado com a organização do código permite que a maior parte das evoluções a realizar sejam alcançadas com razoável facilidade, sendo que, generalizando, a única razão pelas quais elas não foram já realizadas prendeu-se apenas com o factor tempo, tendo sido já imenso feito.

As principais evoluções serão:

- Aumentar Interactividade do Teste – As informações do teste, para os testes explícitos, poderão facilmente ser alteradas para que a informação transmitida ao utilizador seja maior. Poder-se-ia também melhorar a informação visual transmitida pelas barras de progresso. No entanto, esta é uma questão secundária, que pode ser resolvida no âmbito de um *plug-in* de visualização.
- Aumentar Segurança – Neste momento a aplicação comunica com o servidor segundo o protocolo http. A adaptação do código e do servidor para que seja utilizado o protocolo HTTPS impediria a escuta dos pacotes transmitidos. Para esta mudança bastará configurar o servidor para suportar SSL e alterar os métodos respectivos para utilizarem esta tecnologia. Outra mudança seria a de utilizar uma síntese (*hash*) com chave privada, tal como é utilizada na aplicação Java. Note-se que esta questão pode ser implementada já com recurso a um *plug-in* de medição.
- Permitir Usar *Mirrors* – Com um teste de *download* http, é relativamente fácil a utilização de um sistema de *mirrors* tal como é feito na aplicação *web*. Com esta possibilidade em mente as classes já possuem métodos para que seja definido o *mirror* a ser utilizado, mas este é instanciado sempre de forma estática pelo diálogo principal. Esta questão deverá ser facilmente resolvida com a implementação de um plug-in de medição.
- Tamanhos Progressivos – Ainda assumindo que a técnica de medição permanece a de comunicações http, seria plausível efectuar os *downloads* de ficheiros de tamanhos progressivos como é realizado no Applet Java ou então utilizar a informação do contrato para determinar qual o melhor tamanho a ser utilizado. Uma vez mais, um *plug-in* de medição poderá resolver com facilidade esta questão.

Uma das principais conclusões a retirar da implementação relaciona-se com a mudança efectuada entre a primeira e a segunda versões da aplicação. As facilidades retiradas com o uso aprofundado das tecnologias .NET revelou-se bastante compensador, ainda que muitas vezes estas diferenças fossem causa de inexperiência.

O tamanho do ficheiro de instalação, por exemplo, reduziu de 5.5 MB para 420KB.

As diferenças não se reflectiram apenas no desempenho, memória ocupada ou tamanho em disco, como acima de tudo na legibilidade e organização do código.

Como exemplo, vejamos um pequeno excerto das duas versões da aplicação, nas quais se converte uma *String* obtida a partir do registo do Windows para uma variável do tipo inteiro. Em C++, recorrendo-se a múltiplas plataformas, MFC, .NET, e ATL, numas funções utilizam-se uns tipos de *Strings*, noutras, outros. O código resultante acabou por ser o seguinte:

```
teste = rk->GetValue("contractID");
String ^ tempstr = gcnew String("");
tempstr = teste->ToString();
pin_ptr<const wchar_t> wch = PtrToStringChars(tempstr);
size_t origsize = wcslen(wch) + 1;
const size_t newsize = 100;
size_t convertedChars = 0;
char nstring[newsize];
wcstombs_s(&convertedChars, nstring, origsize, wch, _TRUNCATE);
m_intContract = atoi(nstring);
```

O equivalente na segunda versão, em C# .NET, já com o código de leitura do dado a partir do *Registry*:

```
return int.Parse(rk.GetValue("contractID").ToString());
```

Os exemplos são inúmeros.

Esta reorganização do código foi fulcral para permitir a fácil integração das arquitecturas de actualizações automáticas e de extensibilidade por *plug-ins*.

9.3. Auto-Updates

A relevância de uma infra-estrutura deste tipo é enorme, especialmente quando estamos na presença de uma aplicação que interage com outra componente ou aplicação, como é o caso da Estação NETeorológica que deverá ser sempre capaz de comunicar com o servidor central, sem que ambos deixem de evoluir.

Após o lançamento da primeira versão, houve uma quantidade elevada de erros que não puderam ser corrigidos devido ao facto da aplicação já ter sido distribuída. Com a segunda versão, essa questão não se coloca. É possível fazer alterações tanto no servidor como nas aplicações residentes instaladas através de um universo de utilizadores, de forma razoavelmente controlada.

A criação de uma infra-estrutura de actualização automática permitiu desvendar algumas questões internas às aplicações .NET que se revelaram bastante poderosas, nomeadamente a assinatura de *assemblies* e a instanciação de domínios aplicacionais.

As preocupações com a segurança são cada vez maiores, e tal verifica-se na cada vez maior facilidade de implementação de mecanismos que fornecem segurança a uma aplicação, sendo seguro afirmar que mesmo que o servidor central seja comprometido, não é possível esse facto ser usado no sentido de distribuir código malicioso.

As capacidades da infra-estrutura criada cumpriram todos os objectivos definidos, podendo facilmente ser distribuída uma actualização da aplicação, ou esta técnica ser aplicada em outros projectos.

No que diz respeito a evoluções futuras desta técnica, é gratificante referir que os próprios mecanismos de actualização podem ser actualizados automaticamente, já que não deixam de fazer parte da aplicação.

9.4. Plug-ins

A tarefa de implementação de infra-estruturas de *plug-ins* revelou-se mais complexa do que inicialmente se previa.

Efectivamente, o facto de não existirem boas arquitecturas implementadas para este efeito, justifica-se, na medida em que uma arquitectura deste tipo necessita de uma personalização bastante elevada à aplicação em que é utilizada. Isto é visível logo nas decisões associadas à arquitectura candidata, tendo sido identificado um tipo de *plug-in*, o que veio a ser implementado, que apenas se aplica a esta aplicação.

O *plug-in* que, para esta aplicação, se revelava mais útil, apenas diz respeito a esta aplicação e não é genérico ao contrário de um *plug-in* de alterações visuais da interface (*Skins*).

Ainda que o tipo de *plug-ins* a aplicar às aplicações são muitas vezes muito específicos destas, admitindo-se uma determinada metodologia de implementação, foi possível implementar um gestor de *plug-ins* razoavelmente generalista.

A arquitectura gerada pode ser facilmente adaptada na maior parte das situações em que se possa pretender dar-lhe uso. As características principais que o uso desta arquitectura implica são as seguintes:

- A comunicação entre as duas entidades (aplicação e *plug-in*) apenas tem uma classe definida como ponto de contacto, de cada lado. Isto significa que do ponto de vista da aplicação, o *plug-in* é um objecto único e do ponto de vista do *plug-in* a aplicação é também ela um único objecto. De qualquer forma, nada impede que estes objectos

forneçam referências para mais objectos, sendo o número de classes envolvidas virtualmente ilimitado.

- Tem que estar claramente definida a Interface que o *plug-in* deverá implementar. Esta deverá sempre incluir uma função getName() para obtenção do nome do *plug-in*. A definição de uma interface da aplicação é opcional, já que o *plug-in* pode ser utilizado sincronamente, sem poder iniciar contacto com a aplicação.
- Considera-se que os *plug-ins* se encontram numa dada directoria, cada um num ficheiro .dll programado numa das inúmeras linguagens .NET.
- Opcionalmente, pode-se indicar uma chave pública a ser utilizada na importação dos *plug-ins* para verificar a autenticidade dos mesmos.

Outra conclusão a retirar prende-se com a separação entre a implementação da interface e a lógica de medição. Para uma correcta implementação de um sistema de expansibilidade por *plug-ins* é essencial que a separação entre interface e lógica esteja tão clara quanto possível. No entanto, o desenvolvimento em C# não promove essa separação clara, o que dificulta esta tarefa. Por outro lado, quando se pretende uma interactividade mínima na informação fornecida ao utilizador durante o teste, torna-se ainda mais complicada esta separação.

Apesar de se terem identificado três tipos de *plug-ins* a implementar, uma vez que quase todas as possíveis evoluções à aplicação residente, identificadas neste mesmo capítulo, se referem a pequenos aspectos das técnicas de medição, é fácil de concluir que é nesta área que a aplicação tem mais espaço para evolução. Sendo assim, penso que é a única das três vertentes, numa primeira fase na qual a aplicação ainda não se encontra difundida e o potencial de desenvolvimento comunitário ainda é bastante reduzido, cuja implementação poderá colher alguns frutos.

10. Glossário

Aplicação residente: programa que corre localmente, na máquina do utilizador, e realiza medições periódicas da largura de banda do utilizador. Deve estar associada a uma conta de utilizador. Tem por finalidade obter um conjunto relevante de medições ao longo de um período de tempo alargado, para o utilizador interessado saber mais sobre a qualidade da sua ligação.

Applet: ferramenta de medição disponibilizada no contexto de uma página *web*. Encontra-se alojado num dado *mirror*, permite testar a largura de banda em ambos os sentidos: transmissão e recepção. Envia os dados da medição para o servidor central.

Application Manifest: Termo que refere o documento que publica as propriedades actuais de uma aplicação, em especial a sua versão mais recentemente publicada.

Auto-Updates: Funcionalidade de uma aplicação de se actualizar automaticamente, descarregando novas versões de si própria e reiniciando-se de forma mais ou menos transparente para o utilizador.

Banda Larga: o nome usado para definir qualquer conexão acima da velocidade padrão dos modems analógicos (56 *Kbps*). As tecnologias mais difundidas são o ADSL e o cabo.

Bottleneck: Quando aplicado no contexto de redes informáticas, este termo refere a secção, entre dois nós de uma rede, na qual a comunicação entre estes possui um menor throughput.

CLR: *Common Language Runtime*. É a componente da plataforma .NET que executa o código funcionando como uma máquina virtual ou ambiente de execução controlada.

Condições de Utilização: descrição das responsabilidades a que o utilizador e a entidade promotora do sistema NETeorologia se sujeitam ao ser feito um registo por parte do utilizador no sistema.

Contrato: Refere-se ao contrato de prestação de serviço para acesso à Internet, identificado pelo nome do tipo de serviço e respectiva localização. O utilizador faz registo dos seus contratos no sistema, através do sítio *web*, podendo estes ser verificados pelo endereço IP aquando da associação a uma medição (que deve estar conforme ao *IP-Range* definido para o

operador que disponibiliza o contrato). No âmbito do sistema computacional a ser desenvolvido, apenas interessam contratos referentes a acesso em banda larga.

FAQ: registo de perguntas e respostas que representam dúvidas recorrentes por parte dos utilizadores.

Histórico: informação estatística relativa às medições realizadas pelos utilizadores. A apresentação da informação pode ser filtrada de diversas formas: tipo de serviço, localidade, global ou individual (cada utilizador pode consultar apenas as suas próprias estatísticas), tipo de tecnologia (ADSL, cabo, satélite, etc), fonte (*applet* ou aplicação residente).

IP-Range: gama de endereços IP atribuídos a um dado operador. Para cada operador nacional existem um conjunto de gamas exclusivas, normalmente definidas por um endereço de sub rede IP e uma máscara de bits. Permitem identificar o operador que está a fornecer o acesso a um determinado utilizador, bastando para tal conhecer o seu endereço IP.

Largura de Banda: quantidade de informação que pode ser transferida de um nó de rede para outro, durante um determinado intervalo de tempo. Geralmente é usada a unidade de *kbits/s*. No nosso sistema, os valores da largura de banda mostrados são apresentados ao utilizador em *KBytes/s*, unidade de medida habitualmente utilizada pelos programas de transferência de ficheiros (*peer-to-peer*, *download managers* dos *browsers*, etc).

Medição *on-line*: realização de teste de largura de banda através do sítio *web* do sistema. O teste permite medir largura de banda de *downstream*, *upstream* e *ping*. Permite fazer medições de uma forma esporádica, tornando-se assim mais prático para o utilizador ocasional do sistema.

A medição deve conter informação sobre *download*, *upload*, *ping*, data e hora em que foi realizada, e utilizador/contrato a que diz respeito (caso tenha sido feita por utilizador registado).

Mirror: Servidor no qual se encontra alojado o *applet*. Permite ao utilizador escolher a localização a partir da qual pretende fazer o carregamento (para realização de medição *on-line* e periódica).

MVC: O acrónimo inglês refere-se a *Model-View-Controller*. MVC é um padrão de implementação de arquitecturas usado na engenharia de software de aplicações complexas que, geralmente, gerem uma quantidade de dados relativamente elevada. Esta boa prática

indica que o programador deve arquitectar a sua aplicação em três camadas: a do seu modelo de dados (*Model*); a da interface gráfica (*View*) e a da lógica de negócio (*Controller*).

Operador: Entidade que fornece serviço de acesso à Internet através de uma determinada tecnologia (cabo, ADSL, satélite, *dial-up*, etc).

Plug-in: Um *plug-in* é um programa secundário que interage com uma aplicação principal (denominada de *host*), no sentido de estender as suas funcionalidades.

Qualidade de Serviço (QdS): Formalmente, a Qualidade de Serviço, no seu termo em inglês (*Quality of Service*) refere os mecanismos de controlo de uma rede de pacotes, no sentido de providenciar o nível de desempenho solicitado. No entanto, o termo em português é geralmente usado no sentido de indicar uma medição relativa do grau de cumprimento do desempenho contratado a um fornecedor de Internet.

Registo *on-line*: processo que permite ao utilizador anónimo do sistema registar-se, de modo a ter uma conta de utilizador. Esta conta permite-lhe usar a aplicação residente, visto que ela tem de estar associada a um determinado utilizador. Com o registo, o utilizador fornece ao sistema algumas informações obrigatórias (*username* e *password*) e opcionais (nome, localidade, *email*).

SGBD: Sistema de Gestão de Bases de Dados. Refere-se a software encarregue de gerir bases de dados. São geralmente um componente muito importante na arquitectura de uma aplicação orientada a dados. Exemplos de SGBD mais comuns é o MySQL, SQL Server e Oracle.

Throughput: Termo inglês que refere a quantidade de dados digitais por unidade de tempo que são entregues a um terminal numa rede, ou entre dois nós de uma rede. É normalmente medido em bits por segundo (bit/s ou bps).

Tipo de serviço: refere-se à oferta de um determinado de serviço de acesso à Internet disponibilizado por um operador. No âmbito do nosso sistema, é caracterizado pelo valor da largura de banda disponível ao cliente para *downstream* e *upstream*, bem como pela tecnologia de acesso (ADSL, cabo…).

11. Referências

[adobecoldfusion] – "Adobe ColdFusion MX7", http://www.adobe.com/products/coldfusion/

[adobeflash] – "Adobe Flash CS3", http://www.adobe.com/products/flash/

[anacom2005] – ANACOM, "Avaliação do Serviço de Acesso à Internet – Banda Larga: ADSL e Cabo", Outubro 2005.

[bwest] – R. S. Prasad, M. Murray, C. Dovrolis, e K. Claffy, "*Bandwidth estimation: metrics, measurement techniques, and tools*", Novembro 2003.

[cofa2006] – Rúben Correia e Luís Faceira, "Plataforma para medição da qualidade de serviço em tempo real da oferta de banda larga em Portugal", Relatório final da cadeira de Projecto da Licenciatura em Eng. De Computadores e Telemática, Julho 2006.

[ecma262] – European Computer Manufacturers Association, "*ECMAScript Language Specification*", http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf, Dezembro 1999

[ecma334] – European Computer Manufacturers Association, "*C# Specification*", http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf, Julho 2005

[ecma335] – European Computer Manufacturers Association, "*Common Language Infrastructure Specification*", http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-335.pdf, Julho 2005

[ezSQL] – "ezSQL Database Class", http://www.jvmultimedia.com/portal/node/6

[garret2005] – Jesse James Garret, "*Ajax: A New Approach to Web Applications*", http://www.adaptivepath.com/publications/essays/archives/000385.php, Fevereiro 2005

[geoip] – "Maxmind's GeoIP", http://www.maxmind.com/app/ip-location

[hostip] – "*HostIP: Community Geotarget IP Project*", http://www.hostip.info

[hveiga2005] – Hélder Veiga, "*Active traffic monitoring for heterogeneous environments*", Abril 2005

[holmes2006] – James Holmes, "*Struts: The Complete Reference, 2nd Edition*", McGraw-Hill Osborne Media, Agosto 2006

[javase] – "Java Standard Edition", http://java.sun.com/javase/

[javasun] – Sun Microsystems, "Java Applets", http://java.sun.com/docs/books/tutorial/deployment/applet/index.html

[jpgraph] – "PHP Graph Creating Library", http://www.aditus.nu/jpgraph

[kim2001] – Steven Kim, "*Java Web Start - Developing and distributing Java applications for the client side*", http://www.ibm.com/developerworks/java/library/j-webstart/, Setembro 2001

[kuze98] – Michael Kunze, "*Let There be Light - LAMP: Freeware Web Publishing System with Database Support*", c't, Dezembro 1998

[mackenzie2004] – Duncan Mackenzie, "*Using Windows XP Background Intelligent Transfer Service (BITS) with Visual Studio .NET*", http://msdn2.microsoft.com/en-us/library/ms997639.aspx, Fevereiro 2004

[microsoftvb] – "Microsoft Visual Basic .NET", http://msdn2.microsoft.com/en-us/vbasic/

[microsoftjsharp] – "Microsoft Visual J Sharp", http://msdn.microsoft.com/vjsharp/

[odetocode2004] – "*What ASP.NET Programmers Should Know About Application Domains*", OdeToCode, Dezembro 2004

[packets] – C. Dovrolis, P. Ramanathan, e D. Moore, "*What do packet dispersion techniques measure?*", Abril 2001.

[pdarragh2006] – Patrick Darragh, "*ClickOnce: Bringing Ease and Reliability to Smart Client Deployment*", CoDe, Fevereiro 2006

[penry2003] – Andrew Penry, "*A Comparison of two major dynamic web platforms (LAMP vs. WISA)*", http://www.shawnolson.net/a/302/a-comparison-of-two-major-dynamic-web-platforms-lamp-vs-wisa.html, Maio 2003

[phpBB] – "*phpBB: Creating Communities Worldwide*", http//www.phpbb.com

[php_gd] – "PHP: Image Functions", http://www.php.net/gd

[php_session] – "Session Handling Function", http://www.php.net/session

[jdick2004] – Jonathan Dick, "*Plug-ins in C#*", http://www.codeproject.com/csharp/PluginsInCSharp.asp, Março 2004

[rfc 1321] – R. Rivest, "*The MD5 Message-Digest Algorithm*", Abril 1992. RFC1321

[rubyonrails] – "Ruby On Rails", http://www.rubyonrails.org/

[matsumoto2000] - Yukihiro Matsumoto, "*The Ruby Programming Language*", http://www.informit.com/articles/article.asp?p=18225&rl=1, Junho 2000

[seda2004] – Jan Seda, "*Strong Names Explained*", http://www.codeproject.com/dotnet/StrongNameExplained.asp, Novembro 2004

[serverside2005] – "*Application Servers Comparision Matrix*", http://www.theserverside.com/tt/reviews/matrix.tss, Março 2005

[simon2002] – Jonathan Simon, "*A comparison of auto-updating solutions for thick Java clients*", JavaWorld, Novembro 2002

[silverlight2007] – "*Microsoft Unveils Silverlight to Power the Next Generation of Media Experiences on the Web*", http://www.microsoft.com/presspass/press/2007/apr07/04-15WPFEPR.mspx, Abril 2007

[smarty] – "Smarty: Template Engine", http://smarty.php.net

[wikipedia] – "Wikipedia, the free encyclopedia", http://www.wikipedia.org

[zendframework] – "Zend Framework", http://framework.zend.com/

Anexo A – Modelo de Casos de Utilização

Neste anexo serão detalhados os casos de utilização definidos no âmbito da análise de requisitos do sistema.

Para cada caso de utilização identificado, será apresentada uma tabela em que é identificada a finalidade do caso, os actores intervenientes, condições da sua realização, requisitos funcionais que o caso de utilização pode garantir, e uma descrição da sequência prevista de eventos comunicados entre actor e sistema. Poderão também ser apresentados, caso se justifique, requisitos especiais, aspectos em aberto e um diagrama de actividades que ilustre a sequência de eventos.

Efectuar registo

Nome:	**Efectuar Registo**	
Âmbito:	NETeorologia	
Finalidade:	Criação de uma nova conta no sistema, fornecendo informação pessoal, sendo esta identificada por um nome de utilizador e palavra-chave.	
Actores:	Utilizador anónimo	
Pré-condições:	O utilizador não fez *login* no site	
Requisitos funcionais:	1.4; 2.5; 3.8;	
Sequência típica dos eventos:	**Actor** 1. Clica na opção adequada da interface. 3. Caso concorde com as condições, faz o preenchimento dos campos e submete dados.	**Sistema** 2. O sistema mostra ao actor uma nova página com campos de preenchimento: nome (opcional), endereço de e-mail, nome de utilizador, palavra-chave (campo duplo de confirmação). Apresenta as condições gerais para o registo no site. 4. Sistema verifica se todos os campos estão preenchidos correctamente. 5. Mostra uma nova página indicando o sucesso da operação. CaU termina.
Sequências alternativas e extensões:		A1. Se o sistema verificar que ocorreram erros em 4, volta a 2, sinalizando os campos onde houve erros.
Requisitos especiais:	Não mostrar palavra-chave ao ser digitada pelo actor; nome de utilizador não pode existir em duplicado no sistema. Todos os campos são obrigatórios excepto o de nome.	
Aspectos em aberto:	Após o sucesso da operação de registo, é feito um *login* automático? Envio de mensagem de correio electrónico para confirmar registo? Haverá mais campos para registar?	

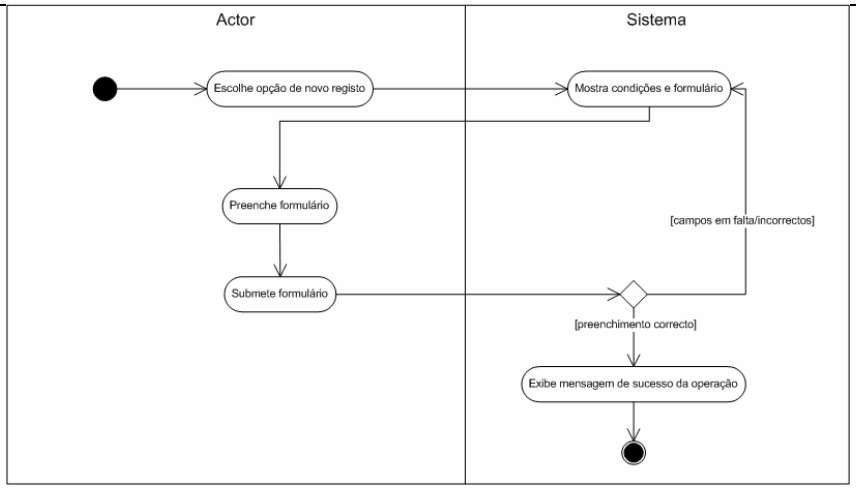


Figura 22 – Diagrama Actividade Efectuar Registo

Efectuar *Login*

Nome:	**Efectuar *Login***	
Âmbito:	NETeorologia	
Finalidade:	O actor identifica-se perante o sistema através do seu nome de utilizador e palavra-chave, de forma a poder usufruir dos serviços disponíveis para utilizadores registados.	
Actores:	Utilizador anónimo	
Pré-condições:	Foi criada anteriormente uma conta para o utilizador que se pretende autenticar.	
Requisitos funcionais:	1.4	
Sequência típica dos eventos:	**Actor** 1. Insere nome de utilizador e palavra-chave, clicando em seguida na opção da interface.	**Sistema** 2. Sistema verifica que dados estão correctos e inicia nova sessão para este utilizador. Termina CaU.
Sequências alternativas e extensões:	A1. Se o utilizador não se lembrar dos dados da sua conta, pode clicar em "Recuperar palavra-chave".	A2. Uma nova *password* é enviada para o endereço de correio electrónico fornecido pelo utilizador, sendo mostrada mensagem com essa informação. Caso não existam registos para esse endereço, o facto é sinalizado. B1. Sistema verifica que dados introduzidos estão incorrectos e mostra mensagem de alerta ao utilizador.
Requisitos especiais:	A comunicação dos dados (nome de utilizador e palavra-chave) deve ser feita de modo seguro, para que os mesmos não sejam vistos por terceiros. Não mostrar palavra-chave ao ser digitada pelo actor.	
Aspectos em aberto:		

Efectuar *Logout*

Nome:	**Efectuar *Logout***	
Âmbito:	NETeorologia	
Finalidade:	O actor indica ao sistema que pretende terminar a sessão como utilizador registado.	
Actores:	Utilizador registado	
Pré-condições:	Existe uma sessão activa para este utilizador.	
Requisitos funcionais:	1.4	
Sequência típica dos eventos:	**Actor** 1. Clica em "*Logout*"	**Sistema** 2. Sistema termina sessão do utilizador e redirecciona para página inicial. Termina CaU
Sequências alternativas e extensões:	(Não tem)	
Requisitos especiais:	(Não tem)	
Aspectos em aberto:	*Auto-logout* após um certo intervalo de tempo?	

Realizar medição *on-line*

Nome:	**Realizar medição *on-line***	
Âmbito:	NETeorologia	
Finalidade:	O utilizador anónimo ou o utilizador registado utilizam a ferramenta *on-line* para efectuar uma medição pontual da largura de banda da ligação do utente.	
Actores:	Utilizador registado, utilizador anónimo	
Pré-condições:	O servidor de onde vai ser feito o carregamento dos ficheiros de teste está activo.	
Requisitos funcionais:	1.1; 1.2	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Escolhe opção adequada da interface.	
		2. Mostra os servidores (*mirrors*) que estão disponíveis para serem usados para o teste
	3. Escolhe o servidor que pretende usar.	
		4. Inicia o carregamento do *applet* de medição a partir do servidor escolhido.
	5. Clica na opção que lhe permite iniciar o teste.	
		6. Applet faz os testes de *download*/*upload* de ficheiros, mostrando barra com estado de evolução do teste.
		7. Após fim do teste, mostra resultados.
		8. Armazena resultados no sistema. CaU termina.
Sequências alternativas e extensões:		A1. Caso verifique que os dados referentes à medição a inserir estão incorrectos, mostra mensagem de erro e não faz o armazenamento dos mesmos.
		B1. Caso seja utilizador registado, mostra lista dos contratos que possa ter.
	B2. Inicia CaU “Associar medição a contrato”.	
Requisitos especiais:	O teste não deve demorar um tempo excessivo, visto que isso não seria de todo do agrado do utilizador. Mecanismo para verificar autenticidade dos dados que são introduzidos, para evitar fraude nos resultados.	
Aspectos em aberto:	Mostrar ocupação de cada servidor? Verificar tempo médio que deve durar o teste. O que fazer se armazenamento dos dados não poder ser feito?	

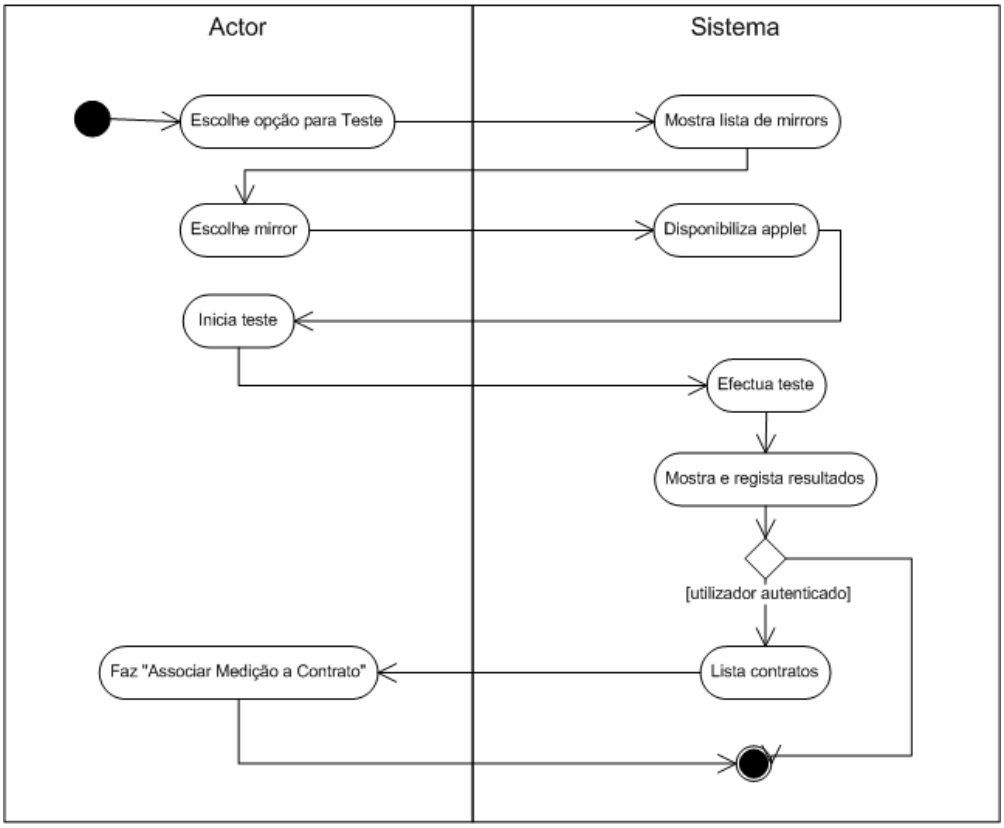


Figura 23 – Diagrama Actividade Realizar Medição *Online*

Associar medição a contrato

Nome:	**Associar medição a contrato**	
Âmbito:	NETeorologia	
Finalidade:	É feita uma associação entre a medição feita pelo utilizador registado e um dos contratos que possui na sua conta.	
Actores:	Utilizador registado	
Pré-condições:	Utilizador encontra-se autenticado no sistema.	
Requisitos funcionais:	1.1; 1.2	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Mostra contratos que constam da conta do utilizador.
	2. Escolhe contrato ao qual pretende associar esta medição. Pode também registar um novo contrato nesta altura, iniciando o CaU “Adicionar contrato”.	
		3. Associa medição ao contrato indicado pelo actor. Mostra mensagem e termina.
Sequências alternativas e extensões:		
Requisitos especiais:	Medição apenas pode ser associada uma única vez.	
Aspectos em aberto:	Mecanismos para garantir que a escolha feita pelo utilizador é confiável.	

Visualizar estatística

Nome:	**Visualizar estatística**	
Âmbito:	NETeorologia	
Finalidade:	O sistema mostra informação tratada estatisticamente de forma gráfica referente às medições, podendo o utilizador escolher entre vários parâmetros de filtragem.	
Actores:	Utilizador registado, Utilizador anónimo	
Pré-condições:	Base de dados onde foi armazenada a informação está em funcionamento.	
Requisitos funcionais:	2.4	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe opção adequada na interface. 3. Escolhe operador e contrato, local (distrito e concelho) sobre o qual quer ver estatísticas. 5. Actor consulta informação. Termina CaU.	**Sistema** 2. Sistema disponibiliza lista de contratos e locais disponíveis. 4. Mostra gráficos relativos às estatísticas do dia e mês actuais para as escolhas pelo actor.
Sequências alternativas e extensões:	A1. Actor pode visualizar outro dia ou mês, ao clicar nas opções adequadas da interface. B1. Se pretender pode visualizar as estatísticas globais ou apenas as suas, escolhendo o filtro adequado na interface (estendendo para "Visualizar estatística de utilizador"). C1. Actor pode alterar o tipo de contrato que está a ver a qualquer altura.	
Requisitos especiais:	Gráficos devem ser facilmente compreensíveis.	
Aspectos em aberto:	Mostrar *upload*/*download*/*ping* em conjunto ou separados? Tipo de gráficos a mostrar?	

Visualizar medições

Nome:	**Visualizar medições**	
Âmbito:	NETeorologia	
Finalidade:	O sistema mostra lista com as últimas medições efectuadas pelo utilizador	
Actores:	Utilizador registado	
Pré-condições:	Base de dados onde foi armazenada a informação está em funcionamento.	
Requisitos funcionais:	2.4	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe opção adequada na interface. 3. Actor consulta informação. Termina CaU.	**Sistema** 2. Sistema apresenta listagem dos registos das medições feitas pelo utilizador, mostrando os seus dados (data, tipo de contrato e sua localização, valor de *download*, *upload* e atraso, bem como a fonte).
Sequências alternativas e extensões:		
Requisitos especiais:	Permitir ao utilizador escolher o número de registos a mostrar por página. Possibilidade de ordenação dos registos.	
Aspectos em aberto:		

Consultar dados de conta

Nome:	**Consultar dados de conta**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador registado faz a consulta dos dados que constam na sua conta.	
Actores:	Utilizador registado	
Pré-condições:	Existe uma conta criada anteriormente para este utilizador; utilizador tem sessão activa para essa conta.	
Requisitos funcionais:	2.7	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe a opção adequada na interface. 3. Actor consulta dados. CaU termina.	**Sistema** 2. Sistema mostra página onde apresenta dados de conta do utilizador.
Sequências alternativas e extensões:	A1. Se actor quiser editar a conta, pode escolher “Editar” na interface, iniciando o CaU “Editar dados de conta”.	
Requisitos especiais:		
Aspectos em aberto:		

Editar dados de conta

Nome:	**Editar dados de conta**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador registado faz a alteração dos dados que constam na sua conta e insere essas alterações no sistema.	
Actores:	Utilizador registado	
Pré-condições:	Existe uma conta criada anteriormente para este utilizador; utilizador tem sessão activa para essa conta.	
Requisitos funcionais:	2.7	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe a opção adequada na interface. 3. Actor altera dados que pretende. 4. Submete dados.	**Sistema** 2. Sistema mostra página com formulário, preenchido com dados de conta já registados. 5. Sistema verifica consistência dos dados submetidos. 6. Mostra nova página com mensagem de sucesso da operação. CaU termina.
Sequências alternativas e extensões:	B1. Se actor quiser registar outro contrato para esta conta, pode fazê-lo ao clicar em “Adicionar Contrato”. C1. Caso pretenda remover um determinado contrato, pode fazê-lo ao clicar em “Remover”, que se encontra ao lado de cada linha de contrato registada.	A1. Em 5, sistema pode encontrar erros nos dados submetidos. Volta a 2, indicando campos onde ocorreram erros.
Requisitos especiais:		
Aspectos em aberto:	Restringir campos que podem ser alterados?	

Adicionar contrato

Nome:	**Adicionar Contrato**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador registado indica ao sistema que pretende associar um novo contrato ao seu registo.	
Actores:	Utilizador registado	
Pré-condições:	Existe uma conta criada anteriormente para este utilizador; utilizador tem sessão activa para essa conta.	
Requisitos funcionais:	2.1	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe a opção adequada na interface. 3. Escolhe contrato que pretende e respectiva localização. 4. Submete escolha.	**Sistema** 2. Sistema mostra nova página com lista de contratos disponíveis, bem como a lista de localizações possíveis. 5. Mostra mensagem de sucesso. CaU termina.
Sequências alternativas e extensões:	(Não tem)	
Requisitos especiais:		
Aspectos em aberto:		

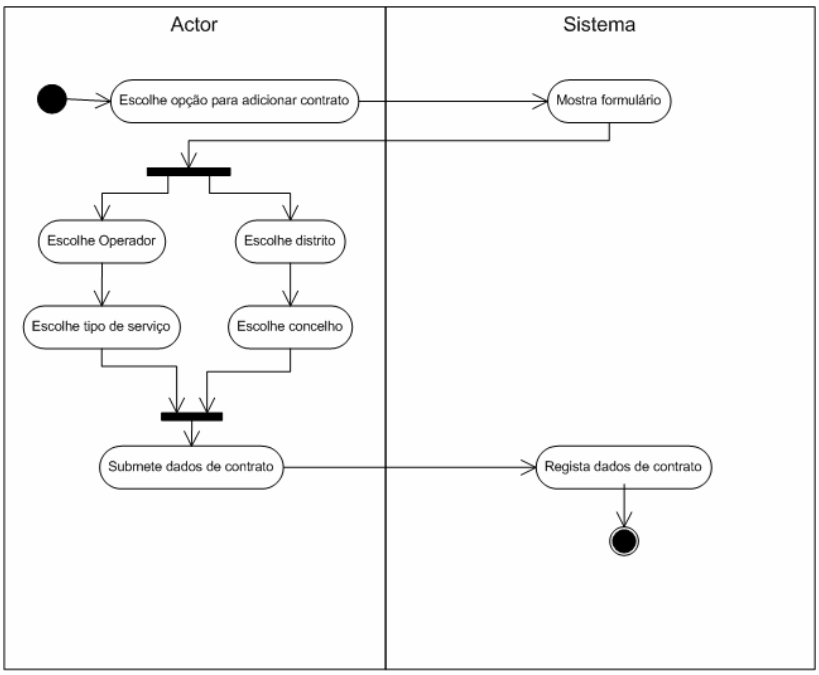


Figura 24 – Diagrama Actividade Adicionar Contrato

Desactivar contrato

Nome:	**Desactivar Contrato**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador registado indica ao sistema que pretende desactivar um contrato, registado para a sua conta.	
Actores:	Utilizador registado	
Pré-condições:	Existe uma conta criada anteriormente para este utilizador; utilizador tem sessão activa para essa conta.	
Requisitos funcionais:	2.1	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe a opção adequada na interface. 3. Actor confirma a desactivação do contrato.	**Sistema** 2. Sistema mostra mensagem a pedir confirmação da desactivação de contrato. 4. Sistema altera registo referente ao contrato em causa, sinalizando que se encontra inactivo. CaU termina.
Sequências alternativas e extensões:	A1. Actor não confirma desactivação de contrato. CaU termina.	
Requisitos especiais:		
Aspectos em aberto:	As medições referentes ao contrato desactivado são apagadas?	

Configurar aplicação

Nome:	**Configurar aplicação**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador altera as opções de funcionamento da aplicação.	
Actores:	Utilizador registado	
Pré-condições:		
Requisitos funcionais:	3.5	
Sequência típica dos eventos:	**Actor** 1. Actor selecciona “Opções” 2. Actor altera as opções que pretende. 3. Clica em “OK” para submeter novas opções.	**Sistema** 2. Sistema mostra janela com opções que podem ser configuradas. 4. Sistema assume novas opções. CaU termina.
Sequências alternativas e extensões:	A1. Actor pretende descartar as opções que fez, clicando em “Cancelar”. CaU termina. B1. Pode escolher “Definir *mirror*” na interface das opções.	
Requisitos especiais:		
Aspectos em aberto:	Que opções existem?	

Consultar ajuda

Nome:	**Consultar ajuda**	
Âmbito:	NETeorologia	
Finalidade:	Utilizador consulta página de ajuda	
Actores:	Utilizador anónimo, utilizador registado	
Pré-condições:		
Requisitos funcionais:	2.1	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe a opção adequada na interface.	**Sistema** 2. Sistema devolve informação de ajuda. CaU termina
Sequências alternativas e extensões:		
Requisitos especiais:		
Aspectos em aberto:	O que consta da ajuda? Possivelmente uma FAQ para permitir aos utilizadores encontrar a resposta ao que pretendem de forma eficaz.	

Listar operadores

Nome:	**Listar operadores**	
Âmbito:	NETeorologia	
Finalidade:	Sistema mostra lista de operadores actualmente registados.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:		
Sequência típica dos eventos:	**Actor** 1. Actor escolhe opção adequada da interface. 3. Actor consulta. CaU termina.	**Sistema** 2. Sistema mostra ecrã com lista dos operadores que estão registados na BD (mostra as suas designações).
Sequências alternativas e extensões:	A1. Actor pode escolher opção "Consultar" para cada operador, iniciando assim o CaU "Consultar dados de operador". B1. Pode igualmente escolher a opção "Editar", que inicia o CaU "Editar operador", para o operador em questão. C1. Pode igualmente escolher a opção "Adicionar Operador", que inicia o CaU "Adicionar operador".	D1. Se verificar que o utilizador não se encontra registado como administrador do sistema, não apresenta a lista e mostra uma mensagem de erro.
Requisitos especiais:	Listagem por ordem alfabética, ordenar por nome	
Aspectos em aberto:		

Adicionar operador

Nome:	**Adicionar operador**	
Âmbito:	NETeorologia	
Finalidade:	O administrador adiciona ao sistema informação referente a um novo operador.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	2.2	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Actor escolhe “Adicionar operador”.	
		2. Sistema mostra ecrã para inserção de dados relativos ao novo operador (nome, morada, *website*, designação, lista *IP-Ranges*).
	3. Insere dados referentes ao novo operador.	
	4. Submete dados.	
		5. Verifica consistência dos dados.
		6. Regista novo operador na base de dados.
		7. Mostra mensagem de sucesso da operação. CaU termina.
Sequências alternativas e extensões:		A1. A verificação encontra erros nos dados que foram submetidos. Volta ao passo 2, indicando erros que ocorreram.
		B1. Se verificar que o utilizador não se encontra registado como administrador do sistema, mostra uma mensagem de erro.
Requisitos especiais:		
Aspectos em aberto:	A informação de *IP-Ranges* deve ser disponibilizado por meio de um ficheiro de texto?	

Consultar dados de operador

Nome:	**Consultar dados de operador**	
Âmbito:	NETeorologia	
Finalidade:	Sistema mostra ao administrador dados relativos a determinado operador	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	2.2	
Sequência típica dos eventos:	**Actor** 1. Actor selecciona, da lista de operadores, qual pretende consultar. 3. Actor consulta informação. CaU termina.	**Sistema** 2. Sistema mostra página de consulta de informações referentes ao operador seleccionado (nome, morada, designação, *website*, *IP-Ranges*, lista de contratos que lhe estão associados).
Sequências alternativas e extensões:	A1. Ao consultar informação, actor pode escolher "Editar Operador", iniciando assim o CaU "Editar dados de operador". B1. Pode igualmente escolher "Editar contrato", que lhe permite iniciar o CaU "Editar tipo de contrato".	C1. Se verificar que o utilizador não se encontra registado como administrador do sistema, não apresenta dados de operador e mostra uma mensagem de erro.
Requisitos especiais:		
Aspectos em aberto:		

Editar dados de operador

Nome:	**Editar dados de operador**	
Âmbito:	NETeorologia	
Finalidade:	Administrador altera os dados relativos a determinado operador	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	2.2	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Actor selecciona a opção adequada na interface.	
		2. Sistema mostra formulário com campos preenchidos pelos dados actualmente constantes no sistema, e a lista de contratos correspondentes ao operador em questão.
	3. Actor muda dados que pretende.	
	4. Submete alterações.	
		5. Verifica consistência de dados.
		6. Regista alterações na BD.
		7. Mostra mensagem de sucesso da operação. Volta a lista de operadores e termina.
Sequências alternativas e extensões:		A1. Se a consistência de dados não for correcta, volta a 2, indicando campos com erro.
	B1. Pode escolher “Adicionar Serviço”, que lhe permite iniciar o CaU “Adicionar tipo de contrato”.	
	C1. Pode optar por “Editar” o tipo de contrato, que lhe permite iniciar o CaU “Editar tipo de contrato”.	
	D1. Ao escolher “Inserir IP Range”, inicia o CaU correspondente.	
	E1. Ao escolher “Remover” para um determinado IP Range, inicia o CaU “Remover IP Range”.	
		F1. Se verificar que o utilizador não se encontra registado como administrador do sistema, não apresenta dados de operador e mostra uma mensagem de erro.
Requisitos especiais:		
Aspectos em aberto:		

Adicionar tipo de contrato

Nome:	**Adicionar tipo de contrato**	
Âmbito:	NETeorologia	
Finalidade:	Administrador adiciona ao sistema um novo tipo de contrato, para que este possa ser escolhidos pelos utilizadores.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	2.2	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Escolhe opção adequada da interface.	
		2. Mostra formulário com campos para serem preenchidos (Nome, Velocidade *Upstream*, Velocidade *Downstream*).
	3. Insere dados do contrato.	
	4. Submete dados.	
		5. Pede confirmação, mostrando dados que foram inseridos.
	6. Confirma.	
		7. Regista novo tipo de contrato, associando-o ao operador. CaU termina.
Sequências alternativas e extensões:	A1. Se actor não confirmar dados em 6, volta a 2.	
		B1. Se verificar que o utilizador não se encontra registado como administrador do sistema, mostra uma mensagem de erro.
Requisitos especiais:		
Aspectos em aberto:		

Editar tipo de contrato

Nome:	**Editar tipo de contrato**	
Âmbito:	NETeorologia	
Finalidade:	Administrador altera parâmetros que caracterizam determinado contrato.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	2.2	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Actor escolhe opção adequada na interface.	
		2. Mostra formulário com campos preenchidos com valores anteriores (Nome, Velocidade *Upstream*, Velocidade *Downstream*).
	3. Altera valor dos campos que pretende (velocidade de *upstream* e/ou *downstream*).	
	4. Submete alterações.	
		5. Pede confirmação, mostrando dados que foram alterados.
	6. Confirma alterações.	
		7. Regista alterações. CaU termina.
Sequências alternativas e extensões:	A1. Actor pode não confirmar alterações. Volta a 2.	
		B1. Se verificar que o utilizador não se encontra registado como administrador do sistema, não apresenta dados de contrato e mostra uma mensagem de erro.
Requisitos especiais:		
Aspectos em aberto:		

Adicionar IP Range

Nome:	**Adicionar IP Range**	
Âmbito:	NETeorologia	
Finalidade:	Administrador associa gama de endereços IP a um determinado operador	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado, operador encontra-se registado no sistema	
Requisitos funcionais:	2.2	
Sequência típica dos eventos:	**Actor** 1. Actor selecciona a opção adequada na interface. 3. Submete dados.	**Sistema** 2. Sistema mostra formulário com campos para preenchimento: endereço IP e máscara de bits 5. Verifica dados 6. Regista alterações na BD. 7. Mostra mensagem de sucesso da operação. Volta a lista de operadores e termina.
Sequências alternativas e extensões:		A1. Se a verificação de dados detectar erros, sinaliza o facto. B1. Se verificar que o utilizador não se encontra registado como administrador do sistema, não apresenta dados de operador e mostra uma mensagem de erro.
Requisitos especiais:	Verificar consistência de dados inseridos	
Aspectos em aberto:		

Remover IP Range

Nome:	**Remover IP Range**	
Âmbito:	NETeorologia	
Finalidade:	Administrador remove gama de endereços IP a um determinado operador	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado, operador encontra-se registado no sistema.	
Requisitos funcionais:	2.2	
Sequência típica dos eventos:	**Actor** 1. Actor selecciona a opção adequada na interface. 3. Clica na opção que lhe permite voltar à página de edição do operador. CaU termina.	**Sistema** 2. Sistema sinaliza remoção de IP range ao operador em questão.
Sequências alternativas e extensões:		
Requisitos especiais:		
Aspectos em aberto:		

Listar utilizadores

Nome:	**Listar utilizadores**	
Âmbito:	NETeorologia	
Finalidade:	Sistema mostra lista de utilizadores actualmente registados.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:		
Sequência típica dos eventos:	**Actor** 1. Actor escolhe opção "Listar utilizadores" da interface.	**Sistema** 2. Sistema mostra ecrã com lista dos utilizadores que estão registados na BD (mostra nome de utilizador e localização). CaU termina.
Sequências alternativas e extensões:	A1. Pode escolher filtrar a lista por nome de utilizador ("*username*") ou nome. B1. Actor pode escolher opção "Bloquear" para cada utilizador que esteja activo, iniciando assim o CaU "Bloquear conta de utilizador". C1. Pode escolher a opção "Desbloquear", para cada utilizador bloqueado, que inicia o CaU "Desbloquear conta de utilizador".	
Requisitos especiais:	Listagem por ordem alfabética por omissão, mostra X entradas por página (opção alterável através da interface).	
Aspectos em aberto:		

Bloquear conta de utilizador

Nome:	**Bloquear conta de utilizador**	
Âmbito:	NETeorologia	
Finalidade:	Conta do utilizador fica com a sinalização de se encontrar bloqueada.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado, utilizador não se encontra bloqueado.	
Requisitos funcionais:	2.3	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Actor selecciona, da lista de utilizadores, qual pretende bloquear, seleccionando “Bloquear”.	
		2. Sistema mostra mensagem a indicar que o utilizador em causa foi bloqueado. CaU termina.
Sequências alternativas e extensões:		
Requisitos especiais:	Se utilizador já estiver bloqueado, assinalar facto.	
Aspectos em aberto:		

Desbloquear conta de utilizador

Nome:	**Desbloquear conta de utilizador**	
Âmbito:	NETeorologia	
Finalidade:	Conta do utilizador fica com a sinalização de se encontrar activa.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado, utilizador encontra-se bloqueado.	
Requisitos funcionais:	2.3	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Actor selecciona, da lista de utilizadores, qual pretende bloquear, seleccionando “Desbloquear”.	
		2. Sistema mostra mensagem a indicar que o utilizador em causa foi desbloqueado. CaU termina.
Sequências alternativas e extensões:		
Requisitos especiais:	Se utilizador já estiver desbloqueado, assinalar facto.	
Aspectos em aberto:		

Listar *mirrors*

Nome:	**Listar *mirrors***	
Âmbito:	NETeorologia	
Finalidade:	Administrador consulta lista com informação referente aos dados dos diversos *mirrors* registados no sistema	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado;	
Requisitos funcionais:	1.5	
Sequência típica dos eventos:	**Actor** 1. Escolhe opção "Listar *Mirrors*" do interface. 3. Consulta informação. CaU termina.	**Sistema** 2. Mostra lista com informação referente a cada *mirror*.
Sequências alternativas e extensões:	A1. Actor pode escolher a opção "Adicionar *Mirror*" da interface, iniciando assim o CaU "Adicionar *Mirror*". B1. Para cada *mirror* listado, pode escolher a opção "Editar", iniciando assim o CaU "Editar *Mirror*".	
Requisitos especiais:	Modo de visualização: mostrar X entradas por página.	
Aspectos em aberto:		

Adicionar *mirror*

Nome:	**Adicionar *mirror***	
Âmbito:	NETeorologia	
Finalidade:	Administrador adiciona ao sistema um novo *mirror*, para que este possa ser usado pelos utilizadores para realizarem medições *on-line*.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado; *mirror* tem alojado o *applet* para medições.	
Requisitos funcionais:	1.5	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Escolhe opção "Adicionar *Mirror*" do interface.	
		2. Mostra formulário com campos referentes aos dados do *mirror* (designação, endereço *web*, capacidade de *upload*/*download*)
	3. Insere dados do *mirror*.	
	4. Submete dados.	5. Pede confirmação, mostrando dados que foram inseridos.
	6. Confirma.	7. Regista novo *mirror* no sistema. CaU termina.
Sequências alternativas e extensões:	A1. Se actor não confirmar dados em 6, volta a 2.	
Requisitos especiais:		
Aspectos em aberto:		

Editar *mirror*

Nome:	**Editar *mirror***	
Âmbito:	NETeorologia	
Finalidade:	Administrador edita os dados de um determinado *mirror*.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado; *mirror* que se pretende editar encontra-se registado no sistema.	
Requisitos funcionais:	1.5	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Escolhe opção adequada do interface.	
		2. Mostra formulário com campos preenchidos com valores anteriores (Nome, Endereço, Comentários, Estado).
	3. Altera valores de campos que pretende.	
	4. Submete alterações.	
		5. Regista alterações.
		6. Mostra mensagem de operação realizada.
Sequências alternativas e extensões:		A1. Em 5, se não conseguir fazer as alterações, mostra mensagem ao utilizador a indicar o facto.
Requisitos especiais:		
Aspectos em aberto:		

Listar notícias

Nome:	**Listar notícias**	
Âmbito:	NETeorologia	
Finalidade:	Sistema mostra lista de notícias actualmente inseridas.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	6.1	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe opção adequada da interface. 3. Actor consulta. CaU termina.	**Sistema** 2. Sistema mostra ecrã com lista das notícias que estão registadas na BD (mostra a data de inserção e título).
Sequências alternativas e extensões:	A1. Actor pode escolher opção “Ver” para cada notícia, iniciando assim o CaU “Consultar notícia”. B1. Pode igualmente escolher a opção “Editar”, que inicia o CaU “Editar notícia”. C1. Ao escolher a opção “Remover”, inicia o CaU “Remover notícia”. D1. Pode igualmente escolher a opção “Adicionar Notícia”, que inicia o CaU “Adicionar notícia”.	E1. Se verificar que o utilizador não se encontra registado como administrador do sistema, não apresenta a lista e mostra uma mensagem de erro.
Requisitos especiais:	Lista ordenada por data	
Aspectos em aberto:		

Adicionar notícia

Nome:	**Adicionar notícia**	
Âmbito:	NETeorologia	
Finalidade:	Fazer a inserção de uma nova notícia no sistema.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	6.1, 6.2	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe opção adequada da interface. 3. Faz a inserção dos dados relativos à notícia.	**Sistema** 2. Sistema mostra ecrã para inserção de dados relativos à notícia (título e corpo). 4. Regista notícia, inserindo no registo a data actual. 5. Mostra mensagem de sucesso. CaU termina.
Sequências alternativas e extensões:		A1. Em 2, caso não tenha sido preenchido algum dos campos, o sistema sinaliza o facto. B1. Caso não tenha conseguido inserir, mostra mensagem a sinalizar o sucedido.
Requisitos especiais:		
Aspectos em aberto:		

Consultar notícia

Nome:	**Consultar notícia**	
Âmbito:	NETeorologia	
Finalidade:	Permite a consulta da notícia pretendida.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	6.2	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe opção adequada da interface.	**Sistema** 2. Sistema mostra ecrã com dados referentes à notícia (data, título e corpo). CaU termina.
Sequências alternativas e extensões:	A1. Actor pode escolher opção “Editar” para cada notícia, iniciando assim o CaU “Editar notícia”.	
Requisitos especiais:		
Aspectos em aberto:		

Editar notícia

Nome:	**Editar notícia**	
Âmbito:	NETeorologia	
Finalidade:	Fazer a alteração dos dados de uma determinada notícia.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	6.2	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe opção adequada da interface. 3. Faz a alteração dos campos que pretende. 4. Submete alterações.	**Sistema** 2. Sistema mostra ecrã para inserção de dados relativos à notícia (título e corpo), preenchidos com os dados anteriormente registados. 5. Regista alterações. 6. Mostra mensagem de sucesso. CaU termina.
Sequências alternativas e extensões:	A1. Se pretender, pode cancelar a operação, clicando na opção que lhe permite voltar à lista de notícias.	B1. Caso não tenha conseguido registar as alterações, mostra mensagem a sinalizar o sucedido.
Requisitos especiais:		
Aspectos em aberto:		

Remover notícia

Nome:	**Remover notícia**	
Âmbito:	NETeorologia	
Finalidade:	É feita a remoção de uma determinada notícia	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	6.2	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe opção adequada da interface. 3. Confirma remoção.	**Sistema** 2. Sistema pede a utilizador para confirmar remoção de notícia. 4. Registo da notícia é removido do sistema. 5. Mostra mensagem de sucesso. CaU termina.
Sequências alternativas e extensões:	A1. Pode preferir não confirmar remoção. CaU termina.	
Requisitos especiais:		
Aspectos em aberto:		

Realizar medição periódica

Nome:	**Realizar medição periódica**	
Âmbito:	NETeorologia	
Finalidade:	Fazer a realização periódica de uma medição por parte da aplicação residente. De forma a ter estatísticas representativas ao longo de um intervalo de tempo alargado.	
Actores:	Tempo	
Pré-condições:	A aplicação está a correr, intervalo de tempo desde a última medição é superior a 30 minutos (acrescido de valor aleatório para evitar sobrecarga dos servidores).	
Requisitos funcionais:	3.1; 3.2	
Sequência típica dos eventos:	**Actor** 1. Sinaliza aplicação de que deve iniciar uma nova medição de largura de banda.	**Sistema** 2. Faz medição da largura de banda. 3. Armazena resultados localmente. 4. Envia resultados para o servidor. CaU termina.
Sequências alternativas e extensões:		A1. Se verificar que dados foram corrompidos, descarta-os. Aplicação avisa utilizador do facto. B1. Se dados estiverem fora do esperado (largura de banda excessiva em relação ao contrato associado à aplicação), dados são descartados e utilizador é avisado da situação. C1. Caso o servidor (*mirror*) escolhido não responda, tentar outros da lista de *mirrors*.
Requisitos especiais:	O teste não deve demorar um tempo excessivo (<2 minutos), visto que o uso da ligação por parte da aplicação deve ser o mais curto possível, para evitar conflitos com normal utilização por parte do utilizador.	
Aspectos em aberto:	Verificar tempo médio que deve durar o teste. Lista de *mirrors* interna da aplicação obtida como?	

Anexo B – Classes PHP da Camada Lógica

Neste anexo pretende-se documentar as variadas classes que definem a camada lógica da aplicação *Web* da plataforma.

Para cada classe são apresentados os métodos relevantes, devidamente descritos.

district.php: Classe responsável pela obtenção de informação referente aos distritos registados na base de dados, comunicando directamente com esta. Possui definido apenas um método:

- getDistricts(amount, offset, extraSQL) – permite obter a lista de distritos, podendo ser definido um limite, caso o *amount* seja superior a 0, bem como um factor de deslocação, caso o *offset* seja também superior a 0. O argumento *extraSQL* é usado para definir código SQL adicional, nomeadamente para filtragem e/ou ordenação.

municipality.php: Classe idêntica à anterior (district.php), sendo neste caso responsável pelos concelhos presentes na base de dados. O método que possui é o seguinte:

- getMunicipalities(amount, offset, extraSQL) – permite obter a lista de concelhos, com funcionalidade e argumentos idênticos ao apresentado no método da classe anterior.

mirror.php: Classe responsável pela informação referentes aos *mirrors*. Possui 5 atributos (*idMirror*, *baseURL*, *name*, *comments* e *state*), que são inicializados com os valores presentes no registo da base de dados, quando é usado o construtor. Os métodos que fazem parte desta classe são os seguintes:

- Mirror(mode, newIdMirror, newBaseURL, newName, newComments, newState*)* – *constructor* da classe. Pode ser usado de duas formas distintas, consoante o valor de *mode*. Caso seja indicado um "*insert*", trata-se da criação de um novo *mirror*, sendo os atributos inicializados com os valores dos argumentos correspondentes, e feita a correspondente inserção na base de dados (se não for possível fazer a inserção, retorna falso). Caso seja indicado o *mode* "*idMirror*", "*baseURL*" ou "*name*", trata-se da obtenção dos dados de um determinado *mirror* já registado na base de dados, sendo os atributos da classe inicializados com os dados previamente registados. Neste caso o campo da pesquisa é correspondente ao *mode*, sendo o valor presente na pesquisa o que tiver sido passado pelo argumento correspondente (respectivamente *newIdMirror*, *newBaseURL* ou *newName*).
- getIdMirror() / getBaseURL() / getName() / getComments() / getState() – funções que retornam os valores dos atributos correspondentes do *mirror*, previamente instanciado correctamente.
- setBaseURL(newBaseURL) / setName(newName) / setComments(newComments) / setState(newState) – funções que alteram os valores do atributos correspondentes da classe, assim como o campo do registo da base de dados do *mirror* em questão.
- getMirrors(amount, offset, extraSQL) – método para obter a lista de *mirrors*, podendo ser definido um limite, caso o *amount* seja superior a 0, bem como um factor de deslocação, caso o *offset* seja também superior a 0. O argumento *extraSQL* é usado para definir código SQL adicional, nomeadamente para filtragem e/ou ordenação.

- countMirrors(extraSQL) – permite obter o número de *mirros* registados na base de dados, podendo ser usado o argumento *extraSQL* para definir um critério de filtragem.

neteouser.php: Classe que tem a responsabilidade pelos registos dos utilizadores. As variáveis que servem de atributo são: idN*eteoUser, email, name, username, pwd, regDate, lastLogin, state* e *trustLevel.* Os métodos que a constituem são:

- NeteoUser(mode, newIdNeteoUser, newEmail, newName, newUsername, newPwd) – construtor da classe. Pode ser usado para inserir os dados de um novo utilizador, se o *mode* for "*insert*", sendo os atributos da classe e os valores dos campos do novo registo preenchidos com os valores passados pelos argumentos. Os restantes atributos, que não constam da lista de argumentos, são campos com valores por defeito. O construtor pode também ser usado para obter um determinado registo, sendo feita uma pesquisa na base de dados pelo campo indicado em *mode* ("*idNeteoUser*" ou "*username*"), tal como no caso do construtor da classe anterior.
- getIdNeteoUser() / getName() / getEmail() / getUsername() / getPwd() / getLastLogin() / getRegDate() / getState() / getTrustLevel() – métodos que permitem obter os valores actuais dos atributos do utilizador em questão (previamente instanciado pelo construtor).
- setName(newName) / setEmail(newEmail) / setPwd(newPwd) / setLastLogin() / setState(newState) / setTrustLevel(mode) – métodos para actualizar os valores do atributo e respectivo campo, conforme o argumento passado. No caso do setLastLogin(), o valor é actualizado para a data e hora actual do servidor. No caso de setTrustLevel(mode), o argumento indica se o *trustLevel* do utilizador deve ser aumentado ou diminuído.
- getUsers(amount, offset, extraSQL) – permite obter a lista de utilizadores registados no sistema, podendo ser definido um limite e/ou factor de deslocação, caso *amount/offset* sejam maiores do que 0. Adicionalmente pode ser usado o argumento *extraSQL* para definir critérios de filtragem ou ordenação.
- addContract(idMunicipality, idServiceType) - método para adicionar um novo contrato ao utilizador, correspondente a um determinado tipo de serviço e localização (concelho), identificados respectivamente por *idMunicipality* e *idServiceType*.
- deactivateContract(idUserServiceType) – permite desactivar um determinado contrato que o utilizador possua, alterando o campo "active" para 0.
- getContracts(amount, offset, extraSQL) – método para listar contratos. O argumento *amount* permite limitar o número de registos a devolver, *offset* permite definir um factor de deslocação, e *extraSQL* deve ser usado para critérios de filtragem (nomeadamente para obter apenas registos referentes a um determinado utilizador) e ordenação.
- countUsers(extraSQL) – obtém número de registos de utilizadores presentes na base de dados. O argumento deve ser usado para filtragem e ordenação.

news.php: Classe que permite gerir os registos das notícias. Possui os atributos *data*, *title* e *body*. Os métodos que foram definidos para esta classe são:

- News(mode, newDate, newTitle, newBody) – construtor da classe. Caso o *mode* escolhido seja "insert", é feita a inserção de um novo registo de notícia na base de dados, a partir dos valores de *newTitle* e *newBody*. Caso o *mode* seja "date", é feita uma pesquisa por notícia que corresponda ao argumento *newDate* passado. Neste caso, é criada uma nova instância, cujos atributos são obtidos a partir do registo que tiver sido encontrado pela pesquisa, caso o tenha sido.
- getDate() / getTitle() / getBody() – métodos que permitem obter os valores actuais dos atributos do objecto.

- setTitle(newTitle) / setBody(newBody) – métodos para modificar os valores dos atributos e respectivos campos do registo guardado na base de dados correspondente à notícia em questão, anteriormente instanciada. Os novos valores são obtidos a partir dos argumentos passados.
- getNews(amount, offset, extraSQL) – método para obtenção dos registos das notícias registadas. Pode ser definido um *amount* para limitar o número de registos, assim como um *offset* desses registos. O argumento *extraSQL* pode usado para filtragens.
- countNews(extraSQL) – permite obter a quantidade de registos de notícias armazenados na base de dados. Pode ser aplicada filtragem segundo um determinado critério a partir do argumento do método.
- deleteNews() – apaga o registo da notícia correspondente ao objecto que invocou o método.

operator.php: Classe responsável pela informação relativa aos operadores. Os atributos que a caracteriza são *idOperator*, *name*, *address*, *website*, *designation* e *state*. A funcionalidade é implementada a partir dos seguintes métodos:

- Operator(mode, newIdOperator, newName, newAddress, newWebsite, newDesignation) – construtor da classe. O *mode* é usado para escolher o tipo de construtor. Caso seja "insert", trata-se da inserção de um novo registo de operador, com os campos e respectivos atributos inicializados com os valores passados pelos argumentos correspondentes (o campo *state* é inicializado por omissão a "NORMAL"). Caso seja "idoperator" ou "name", é instanciado um novo objecto que corresponde ao registo identificado pelo campo em questão (se existir), cujo valor é passado pelo argumento (*newIdOperator* ou *newName*).
- getIdOperator() / getName() / getAddress() / getWebsite() / getDesignation() / getState() – métodos que possibilitam a obtenção dos respectivos valores dos atributos do objecto.
- setState(newState) / setName(newName) / setAddress(newAddress) / setWebsite(newWebsite) / setDesignation(newDesignation) / - permitem actualizar os valores dos campos do registo em questão e respectivos atributos do objecto relacionado.
- getOperators(amount, offset, extraSQL) – método usado para retornar os registos de operador existentes na base de dados. Podem ser limitados em número através do uso do argumento *amount*, pode ser definido um factor de deslocação (*offset*). Para filtragem e ordenação, pode ser usado o argumento *extraSQL*.
- countOperators(extraSQL) – retorna a quantidade de operadores registados no sistema. Pode ser aplicado um critério de filtragem, recorrendo ao argumento *extraSQL*.
- addIpRange(newIP, newBitMask) – permite adicionar uma nova gama de endereços IP ao operador. Esta gama de endereços é caracterizada por um endereço de subrede *newIP* e por uma máscara de bits *newBitMask*.
- removeIpRange(oldIp) – usado para remover uma determinada gama de endereços IP ao operador. O registo de IpRange em questão é identificado pelo endereço de subrede *oldIp*.
- getIpRanges(amount, offset) – método que retorna os registos de IpRange que pertencem ao operador em questão. O retorno pode ser limitado em número através do *amount*, podendo também ser aplicado um *offset* a esse conjunto de registos.
- getIdOperatorbyIp(Ip) – método que retorna o identificador de operador (idOperator) para o qual o endereço *Ip* passado pertence à sua gama de endereços.

servicetype.php: Classe responsável pela gestão de registos que contêm informação sobre os tipos de serviço disponibilizados por cada operador. Os atributos da classe são *idServiceType*, *refIdOperator*, *name*, *upStream*, *downStream* e *state*. Para se atingir a funcionalidade pretendida, foram ainda definidos os seguintes métodos:

- ServiceType(mode, newIdServiceType, newRefIdOperator, newName, newUpStream, newDownStream) – construtor da classe. O *mode* é usado para escolher o tipo de construtor. Caso seja "insert", trata-se da inserção de um novo registo de tipo de serviço, com os campos e respectivos atributos inicializados com os valores passados pelos argumentos correspondentes (o campo *state* é inicializado por omissão a "NORMAL"). Caso seja "idservicetype" ou "name", é instanciado um novo objecto que corresponde ao registo identificado pelo campo em questão (se existir), cujo valor é passado pelo argumento (*newIdServiceType* ou *newName*).
- getIdServiceType() / getRefIdOperator() / getName() / getUpStream() / getDownStream() / getState() – permitem conhecer os valores dos atributos actualmente definidos para o objecto.
- setName(newName) / setUpStream(newUpStream) / setState(newState) / setDownStream(newDownStream) – métodos para alterar os valores correspondentes dos campos do registo de tipo de serviço armazenado na base de dados, e também os atributos do objecto. Os novos valores desses campos/atributos são passados no argumento de cada método.
- getServiceTypes(amount, offset, extraSQL) – retorna os registos de tipos de serviço presentes na base de dados. O seu número pode ser limitado por *amount* e pode igualmente ser aplicado um *offset* a esse conjunto de registos. Para filtragens e ordenação deve ser usado o argumento *extraSQL*.
- countServiceTypes(extraSQL) – método que devolve a quantidade de registos de tipo de serviço que existem no sistema. Pode ser aplicado um critério de filtragem através de *extraSQL*.

measurement.php: Classe que tem a responsabilidade de gerir os registos relacionados com as medições. Os atributos que a constituem são *idMeasurement*, *time*, *ip*, *download*, *upload*, *delay*, *source*, *version*, *trusted*, *refIdNeteoUser* e *refIdServiceType*. Fazem igualmente parte da sua implementação os seguintes métodos:

- Measurement(mode, newIdMeasurement, newTime, newIp, newDownload, newUpload, newDelay, newSource, newVersion, newRefIdUserServiceType, newRefIdNeteoUser, newRefIdMirror) – constructor da classe. O *mode* indica o modo como o constructor deve actuar. Se for "insert", trata-se da inserção de um novo registo na tabela Measurement, com os valores passados em argumento (o valor do atributo/campo *time* é calculado pelo servidor). Neste caso poderá também ser inserido um registo na tabela MirrorMeasurement, identificando o mirror em que foi feita a medição, se tiver sido realizada através do applet. Se for "ipTime" ou "idMeasurement" instancia um novo objecto a partir de um registo que possua o *newIp* e *newTime* ou o *newIdMeasurement* indicados pelos argumentos respectivos.
- getIdMeasurement() / getTime() / getIp() / getDowload() / getUpload() / getDelay() / getSource() / getVersion() / getTrusted() / getRefIdNeteoUser() / getRefIdUserServiceType() – métodos para obter os valores do diversos atributos do objecto.
- removeMeasurement() – procede à eliminação do registo relativo ao objecto em causa.
- getMeasurements(amount, offset, extraSQL) – devolve um conjunto de campos relacionados com cada medição, que incluem: todos os campos da tabela Measurement; "downP" (percentual do download em relação ao definido para o tipo de serviço associado a esta medição); "upP" (percentual idêntico ao anterior, para o upload); eventual referência do identificador de mirror, o estado do utilizador que a realizou, identificador do contrato, do tipo de serviço, do concelho e do distrito. Podem ser limitados em número pelo *amount* e ou ser aplicado um factor de deslocação –

offset. Também pode ser aplicado um critério de filtragem ou de ordenação, sendo para isso usado o argumento *extraSQL.*

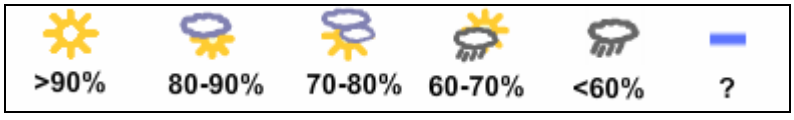
- countMeasurements(extraSQL) – devolve a quantidade de registos de medições, que podem ser filtrados através de *extraSQL*.
- getUserMeasurements(amount, offset, idNeteoUser, extraSQL) – devolve informação relacionada com as medições de um determinado utilizador, identificado por *idNeteoUser*. Os campos devolvidos são *time*, *download*, *upload*, *delay*, *source*, *STName* (nome do tipo de serviço) e *MName* (nome do concelho). Aos registos obtidos pode ser aplicado um *amount* que limita a sua quantidade, um *offset* e *extraSQL* para filtragem e ordenação.
- getConnectionState(amount, idNeteoUser) – para cada medição confiável (valor de *trusted* a 1), retorna o percentual de download e upload em relação ao contrato declarado, bem como o valor do atraso (*delay*), para o utilizador identificado por *idNeteoUser*. Os registos obtidos são referentes aos últimos 30 dias, ordenados descrescentemente, podendo ser ainda limitados em quantidade através de *amount*.
- getDistrictState(amount, idDistrict) – método com funcionalidade idêntica ao anterior, neste caso referente a um determinado distrito identificado por idDistrict e aos últimos 7 dias.
- setTrusted(trust) – estabelece o grau de confiança da medição (“trusted” ou “untrusted”), consoante o valor de *trust*.
- setRefIdNeteoUser(newRefIdNeteoUser) – permite fazer a associação à medição a um determinado utilizador, identificado pelo argumento. setRefIdUserServiceType(newRefIdUST) – método que associa ao registo da medição um determinado tipo de contrato, que existe para um dado utilizador, identificado pelo argumento.

session.php – não se trata de uma classe como os casos anteriores, mas antes um módulo de auxílio à gestão das sessões. Como base são utilizadas as facilidades de gestão de sessões embutidas em PHP [php_session], disponíveis a partir da versão 4.0 dessa linguagem de programação.

painter.inc – módulo responsável pela criação da imagem apresentada na página inicial, que consiste no mapa dinâmico com os estados das ligações das diversas regiões do país.

Em termos gerais, o seu funcionamento consiste no seguinte: após criar a imagem base do mapa, para cada uma das doze regiões presentes, obtém o estado actual da qualidade a partir de uma variável global onde estão guardados esses valores; em seguida imprime na imagem base uma imagem correspondente ao estado, nas coordenadas que estão definidas para essa região; no fim de todas as regiões terem sido impressas no mapa, faz a criação da imagem, com o tipo PNG.

Tendo em conta o estado da região do mapa, que consiste num valor entre 0 e 100% ou eventualmente nulo, é escolhida um dos seguintes ícones para impressão:



Para estas operações foram usadas funções que fazem parte da biblioteca GD [gd], que permite estender a funcionalidade do PHP de modo a ser possível a criação e manipulação de imagens.

Anexo C – *Scripts* PHP da Camada de Visualização

De seguida são apresentados os diversos documentos que contêm o código PHP referente às páginas do *website*. sendo explicada: a sua funcionalidade, através de uma explicação genérica das operações que realiza; os módulos/classes que usam para servir de suporte a essa funcionalidade; o(s) *template(s)* de que se servem para criar as páginas que são exibidas aos utilizadores. É feita ainda uma referência ao Caso de Utilização que é concretizado pela página em questão.

De referir ainda a existência de dois elementos, que são usados ao longo de todo o *website* pelas várias páginas: o cabeçalho e o rodapé. Uma explicação da sua função e do seu funcionamento pode ser consultada no fim desta lista.

Caso haja necessidade de uma compreensão mais profunda e fundamentada das páginas que compõem o *website*, deve ser consultado o próprio código PHP que consta dos ficheiros, o qual é acompanhado por comentários que ajudam nessa compreensão.

index.php – página de entrada do *website*. Mostra mapa com a qualidade da banda larga medida nas várias regiões do país, bem como da qualidade da ligação do utilizador, se estiver registado e tiver medições recentes. Apresenta também as notícias actuais.

- Funcionalidade – No topo (logo abaixo do cabeçalho) é mostrado um mapa com o estado da qualidade da banda larga, dividido por regiões. As regiões podem englobar entre 1 e 3 distritos (ou Região Autónoma, no caso dos Açores e da Madeira), que sejam adjacentes, de modo a organizar a forma como a informação é apresentada ao utilizador. Ao passar com o ponteiro do rato por cima do ícone de cada região surge uma "tooltip" a indicar os nomes dos distritos que fazem parte dessa região e o estado da região, em percentagem, caso existam medições para esses distritos. Esta percentagem é calculada da seguinte forma: para cada distrito, obtém os percentuais dos valores de *download* e *upload* das medições em relação ao contrato declarado para essa medição, fazendo em seguida a média desses valores; para cada região faz a média dos valores de cada distrito; por último, o estado da região é calculado a partir dessas duas médias, sendo que à média de *download* é atribuído um peso maior do que à média de *upload*, na proporção 4 para 1. Foi decidido dar uma maior proporção à média de *download* visto que para o utilizador ela tem uma maior preponderância na sua percepção da qualidade da sua ligação, dado ser o sentido preferencial em que são feitas as comunicações dos clientes residenciais.

  O mapa mostra ainda o estado da ligação do utilizador, se este estiver registado e possuir medições recentes, sendo este estado calculado de maneira similar ao feito para as diversas regiões.

  De referir ainda que são apresentadas as notícias mais recentes.
- Classes/módulos – é usada a classe *NeteoUser* para obter dados relativos ao utilizador autenticado; a classe *Measurement* para obter os dados das medições para criar o mapa dinâmico; a classe *News* para os dados das diversas notícias a apresentar na página; o módulo *Smarty* para interligação com os *templates* e o módulo *Session* para obter informações sobre sessões activas que possam existir.
- Template – é usado o *template* "index.tpl". De referir que o *template* obtém a imagem do mapa a partir de um pedido que faz ao módulo drawMap.php, passando-lhe os estados das várias regiões do mapa. Este módulo obtém uma imagem do mapa em formato PNG que corresponde ao conjunto de estados que recebe, através do uso do módulo painter.inc, passando a este os dados necessários para ele conhecer o estado das regiões do mapa.

  Para o administrador, o interface permite escolher opções que lhe permitem "Listar Notícias", "Adicionar Notícia", "Editar" e "Remover" para cada notícia, o que corresponde a iniciar os Casos de Utilização correspondentes.

History.php – esta página mostra os gráficos relativos aos valores de *download*, *upload* e atraso RTT das medições registadas na base de dados, correspondendo ao Caso de Utilização "Visualizar Estatísticas". Permite a aplicação de filtros às estatísticas a mostrar, podendo ser escolhido o distrito e concelho, o operador e tipo de serviço, a fonte da medição, bem como o dia ou mês que se pretende consultar.

- Funcionalidade – os principais elementos da página são os dois gráficos que apresentam as estatísticas para o dia e mês escolhido, que por omissão correspondem ao dia e mês actuais. Para a criação destes gráficos, é necessário obter os dados relativos às medições que correspondam aos critérios de filtragem definidos pelo utilizador. Após isso, são calculados os valores médios dessas medições para intervalos de 15 minutos, a serem mostrados no gráfico diário, e para intervalos diários, a serem mostrados no gráfico mensal……
- Classes/módulos – são usadas as classes *District*, *Municipality*, *Operator* e *ServiceType* para obter dados relativos a distritos, concelhos, operadores e respectivos tipos de serviço, para constarem nas caixas de selecção que permitem ao utilizador definir critérios de filtragem; é usado o módulo *Session* para verificação de sessões activas, o que permite escolher a visualização das medições do utilizador autenticado; é usado o módulo *Smarty*.
- Template – o *template* usado é "History.tpl". De modo idêntico ao descrito para a criação do mapa dinâmico, os gráficos são obtidos através da chamada de um módulo – "GraphGenerator.php" – ao qual são passados os dados que caracterizam as estatísticas que se pretendem visualizar (os critérios de filtragem). Este módulo fica responsável pela obtenção dos dados das medições (através do uso da classe *Measurement*), fornecendo-os em seguida ao gerador de gráficos, definido pelo módulo "jgraph.php" e "jgraph_line.php".

  Para o utilizador autenticado, a interface apresenta uma opção que permite escolher "Minhas Medições", o que possibilita o início do Caso de Utilização "Visualizar medições".

ListMirrors.php – página que permite ao utilizador escolher um *mirror* a partir do qual pode realizar um teste de qualidade da sua ligação, através do applet, permitindo iniciar o Caso de Utilização "Realizar medição online". Pode igualmente proceder ao descarregamento do instalador da aplicação de medição residente.

- Funcionalidade – para a criação desta página, é preciso listar os *mirrors* que se encontram registados no sistema e que permitem fazer testes esporádicos a partir do applet.
- Classes/módulos – é utilizada a classe *Mirror*, que permite obter os dados dos *mirrors* registados na base de dados; são ainda usado o módulo *Session* para gestão das sessões e *Smarty*, para ligação ao *template*.
- Template – o *template* usado é "ListMirrors.tpl". Este possui definida uma secção *Smarty* (identificável pela etiqueta *{section}*) para fazer a construção da tabela, de forma automática, a partir dos dados referentes aos diversos registos de *mirrors* que lhe são passados.

  Caso se trate de um administrador a visualizar a página, o interface permite-lhe "Adicionar *Mirror*" ou "Editar" o *mirror*, iniciando os Casos de Utilização relacionados.

EditMirror.php – página que permite a edição de um mirror existente no sistema ou o registo de um novo mirror, o que corresponde à realização dos Casos de Utilização "Editar mirror" e "Adicionar Mirror".

- Funcionalidade – a página verifica se é um administrador que está a aceder.

No caso de ser a edição de um registo, são obtidos os dados do *mirror*, a partir do identificador passado em parâmetro, para serem apresentados ao utilizador. É ainda feita uma verificação para garantir que o identificador se refere a um *mirror* que exista.

- Classes/módulos – além dos módulos habituais *Session* e *Smarty*, é necessário o uso da classe *Mirror*, para efectuar a verificação da existência do *mirror*, a inserção de um novo registo ou a alteração dos campos do mirror.
- Template – é usado o *template* "EditMirror.tpl", que apresenta um formulário, que se encontra preenchido caso seja a edição de um *mirror*. Os campos do formulário são verificados por código Javascript aquando da sua submissão, que é feita para a página "InsertMirror.php".

InsertMirror.php – página para inserção dos dados de um registo de mirror (novo registo ou edição de registo já existente), recebidos por submissão do formulário da página anterior.

- Funcionalidade – a página faz a distinção entre novo registo e edição de um já mirror já registado.

  Para a primeira situação, verifica se os campos obrigatórios foram passados, confirma se o nome do novo mirror não existe ainda na base de dados e, caso tudo esteja correcto, faz a inserção do novo registo de *mirror*.

  No caso da edição, verifica se o *mirror* em questão existe e em caso afirmativo, actualizando o registo conforme os dados que foram passados à página.

  Em ambos os casos, faz a passagem dos valores para o *template*, de modo a serem apresentados ao administrador.
- Classes/módulos – é utilizada a classe *Mirror*, para realizar as operações necessárias para verificar os registos de mirrors presentes na base de dados, inserção de um novo registo e alteração de campos de um registo já existente.

EditUser.php – página para registo de um novo utilizador ou consulta/edição da conta de utilizador já registado no sistema. Corresponde respectivamente ao início do Caso de Utilização "Efectuar Registo", "Consultar Dados de conta" ou "Editar dados de conta".

No caso da edição, são ainda disponibilizadas opções que permitem adicionar um novo tipo de contrato, bem como a desactivação de contratos já registados pelo utilizador (concretizando os Casos de Utilização "Adicionar contrato" e "Desactivar contrato"):

- Funcionalidade – a página reage de modo diferente consoante haja uma sessão de um utilizador activa. Caso haja, é feita a pesquisa de dados relativos ao utilizador em questão, que são apresentados, quer para consulta quer para edição, sendo esta diferença indicada por um parâmetro passado à página. São mostrados os dados da conta do utilizador, assim como eventuais contratos que tenha registado (que podem ser identificados pelo tipo de serviço e concelho a que dizem respeito).
- Classes/módulos – esta página faz uso da classe *NeteoUser*, para obter os dados da conta do utilizador se houver alguma sessão activa, o que é verificado com recurso ao módulo *Session*. Para usar os *templates*, é usado o módulo *Smarty*.
- Template – podem ser usados dois *templates* para esta página em questão: "ViewUser.tpl" e "EditUser.tpl". O primeiro permite fazer a consulta dos dados da conta do utilizador, oferecendo ainda opções no interface que permitem que o utilizador inicie os Casos de Utilização "Editar dados de conta" ou "Visualizar Medições", e também opções para listar operadores, utilizadores, notícias e mirrors, caso se trate de um administrador a consultar a página.

  O segundo contém um formulário, que se apresenta em branco se for um novo registo ou preenchido se se tratar da edição de dados de uma conta existente. No primeiro caso é ainda mostrada uma mensagem relativa ao processo de registo no *website*.

De referir ainda em relação ao formulário que, aquando da sua submissão, são feitas algumas verificações dos campos através do uso de Javascript. A submissão é feita para a página "InsertUser.php".

InsertUser.php – página de inserção de dados de registo (novo ou edição de um registo já existente), que recebe os dados provenientes do formulário preenchido na página "EditUser.php".

- Funcionalidade – a página recebe os parâmetros do formulário. Para o caso de um novo registo, verifica se os campos obrigatórios (*username*, *email* e *pwd*) estão preenchidos e se não existe nenhum registo na base de dados com esse *username*. Nesta situação, além da inserção dos dados é ainda feito o login automático para o utilizador que se registou, de modo a este poder começar a usufruir das funcionalidades reservadas para os utilizadores autenticados.
- Classes/módulos – esta página faz uso dos módulos *Smarty* e *Session*, e da classe *NeteoUser*, que permite fazer as operações de verificação de *username*, de inserção de novo registo e de alteração de campos de um dado registo..
- Template – é usado o *template* "InsertUser.tpl", que faz o login automático. Este login é feito através de um formulário, com os campos invisíveis ao utilizador e preenchidos com as variáveis passadas, que é submetido assim que é carregada a página. Após esta operação, imperceptível ao utilizador, é então mostrada a mesma página, desta vez apresentando uma mensagem de boas-vindas, no caso de um novo registo, ou uma curta mensagem a indicar o sucesso da operação de edição do registo.

AddContract.php – página que permite fazer a adição de um novo contrato à conta do utilizador, realizando assim o Caso de Utilização "Adicionar contrato". O utilizador define o seu novo contrato através da escolha de um concelho e de um tipo de serviço oferecido por um operador.

- Funcionalidade – a página é usada para fazer a escolha de contrato/localização, assim como a própria inserção do novo contrato, sendo esta distinção feita através dos parâmetros passados.

  No caso da escolha, são pesquisados os registos referentes aos distritos (e estando um distrito escolhido, os registos referentes aos concelhos desse distrito) e os registos de operadores (e se estiver um operador escolhido, são também obtidos os registos dos tipos de serviço desse operador). Esses são os dados necessários para que o utilizador defina o seu novo contrato.

  Após a escolha, o utilizador faz a submissão dos dados, para esta mesma página. Nessa altura, são feitas as verificações para garantir que os dados são válidos, sendo em seguida verificado se o utilizador já tinha registado anteriormente este contrato (e desactivado entretanto), caso em que procede à sua reactivação. Em opção faz a inserção normal de um novo contrato ao utilizador.

  De referir ainda que, se este novo contrato for criado no contexto de associação a medição, após a inserção do contrato, procede à associação.
- Classes/módulos – São utilizadas diversas classes para se obter a funcionalidade desta página: *District* para obter a lista de distritos; *Measurement* para verificações de segurança e actualização do tipo de contrato associado; Municipality para obter a lista de concelhos do distrito escolhido; *NeteoUser* para fazer a verificação e inserção de um novo contrato ao utilizador; *Operator* para aceder aos registos de operadores existentes; *ServiceType* para listar os tipos de serviço disponibilizados pelo operador escolhido.
- Template – é utilizado o *template* "AddContract.tpl" para escolha do tipo de serviço e localização correspondentes ao novo contrato. Após a submissão desses dados, é

usado o *template* "InsertContract.tpl", que apenas mostra uma mensagem a indicar o estado da operação de inserção de um novo contrato e uma ligação para aceder à página de edição da conta do utilizador.

Help.php – apresenta uma página de ajuda, com um formato de FAQ.

- Funcionalidade – trata-se de uma página essencialmente estática, não apresentando nenhuma funcionalidade que requeira uma descrição detalhada.
- Classes/módulos – faz uso do módulo *Session*, para garantir a continuação de eventuais sessões do utilizador, e do módulo *Smarty*, para o *template*.
- Template – o template usado é "Help.tpl". Trata-se de uma página essencialmente estática, sendo apenas de referir a inclusão, recorrente em todas as páginas, do cabeçalho e do rodapé.

ListOperators.php – Página responsável pela apresentação dos operadores registados no sistema, sob a forma de uma tabela, correspondendo ao Caso de Utilização "Listar Operadores". São disponibilizadas opções para adicionar um novo registo de operador, bem como para consultar em pormenor ou editar os dados de cada operador listado (correspondendo a realizar os Casos de Utilização "Adicionar Operador", "Consultar dados de operador" e "Editar dados de operador").

- Funcionalidade – a página verifica se é um administrador que a está a consultar, visto que apenas este tipo de utilizador pode aceder a ela, caso contrário mostra mensagem de erro. Após isto, obtém os dados dos diversos operadores que estão registados na base de dados. Em seguida realiza alguns cálculos para definir os registos a mostrar e eventualmente as ligações para a página anterior ou seguinte, visto que apenas mostra uma certa quantidade deles por página. De referir ainda que os registos podem ser ordenados por *nome*, de forma ascendente ou descendente.
- Classes/módulos – além dos módulos habituais (*Session* e *Smarty*), é usada a classe Operator, que permite a obtenção do conjunto de registos que se pretendem exibir na página.
- Template – é usado o template "ListOperators.tpl", que consiste essencialmente numa tabela (criada a partir de uma *{section}*) onde são exibidos para cada operador os seus dados (nome, *website*, designação, estado), bem como duas opções, uma para consultar dados com mais pormenor, outra para editar esses dados. Podem ainda existir ligações para a página anterior ou seguinte, conforme ainda existam mais registos a consultar. De referir ainda que o campo *nome* do cabeçalho permite fazer uma ordenação ascendente ou descendente.

EditOperator.php – página que permite adicionar um novo operador, consultar ou editar os dados de um operador já registado no sistema, permitindo a realização dos Casos de Utilização "Adicionar Operador", "Consultar dados de operador" e "Editar dados de operador".

- Funcionalidade – a página apresenta-se de modo diferente, consoante se trate do registo de um novo operador ou da consulta/edição de dados de um operador já registado, sendo estas diferenças distinguidas por parâmetros passados à página.

  No caso da consulta/edição, é feita uma pesquisa para obter os dados do operador em causa, incluindo os que se referem às gamas de endereços IP bem como aos tipos de serviço disponibilizados por este operador.

- Classes/módulos – esta página faz uso da classe *Operator*, que lhe permite retornar os dados do operador, quando se trata da consulta/edição. Faz igualmente uso dos módulos *Session* e *Smarty*.
- Template – os *templates* a usar podem ser "ViewOperator.tpl" ou "EditOperator.tpl".

  O primeiro é usado para mostrar a página de consulta, onde são exibidos os dados do operador, as suas gamas de endereços IP e os seus tipos de serviço.

  O segundo apresenta um formulário para preencher com os dados do operador, que se encontra em branco se for um novo registo ou preenchido se for a edição de um registo já existente na base de dados.

  De referir que os campos do formulário são verificados com recurso a código Javascript, sendo os dados enviados para a página "InsertOperator.php" aquando da submissão.

  No caso da edição, a página mostrada ao administrador contém ainda: opções para inserir uma gama de endereços IP (para realização do Caso de Utilização "Adicionar IP Range"; a lista de endereços já registados, com opção para remover (que realiza o Caso de Utilização "Remover IP Range"); opção para adicionar um novo tipo de serviço (iniciando assim o Caso de Utilização "Adicionar tipo de contrato"); a lista de tipos de serviço, com a opção de edição de cada um (correspondente ao Caso de Utilização "Editar tipo de contrato").

InsertOperator.php – esta página tem a função de receber os dados do formulário que consta da página anterior ("EditOperator.php") e de, após ter feito algumas verificações, os registar na base de dados.

- Funcionalidade – é feita uma verificação, através da análise dos parâmetros passados, que permite distinguir entre uma nova inserção e a edição de um registo. Se se tratar de uma inserção, verifica que todos os campos estão preenchidos e se estiverem, faz a inserção de um novo registo de operador no sistema, após verificar que ainda não existe nenhum operador com o mesmo nome. Tratando-se de uma edição, instancia um novo *Operator* correspondente ao identificador passado em parâmetro, alterando em seguida os campos que forem precisos.
- Classes/módulos – além dos módulos usuais para controlo de sessões e interligação com os *templates*, é usada a classe *Operator* para realizar as operações de inserção de um novo registo de operador, de alteração dos valores dos campos do registo e verificação de operadores.
- Template – o *template* para a página é "InsertOperator.tpl", que apresenta os dados que foram inseridos/alterados, mostrando uma mensagem de sucesso (caso não tenham havido erros).

EditServiceType.php – página que permite registar um novo tipo de serviço ou a edição de um que já esteja registado no *website*.

- Funcionalidade – é feita a verificação do tipo de utilizador que está a consultar a página, sendo apenas permitido o acesso a administrador. Os parâmetros passados são verificados para decidir se é uma nova inserção ou a edição de um registo já existente. No primeiro caso deve ser passado um identificador do operador para o qual vai ser registado este tipo de serviço, enquanto que no segundo caso é passado o identificador do tipo de serviço a editar, sendo em seguida obtidos os dados desse operador para serem apresentados ao utilizador.
- Classes/módulos – além dos módulos habituais de *Session* e *Smarty*, é feito o uso das classes *ServiceType* e *Operator*. A primeira permite obter os dados do tipo de serviço

que se pretende consultar. A segunda é usada para verificar que a referência identificativa do operador é correcta.

- Template – o *template* usado é "EditServiceType.tpl", o qual possui um formulário, que pode estar preenchido ou em branco conforme se trate de uma edição ou de um novo registo. Os campos do formulário são verificados por código Javascript aquando da sua submissão.

InsertServiceType.php – página responsável pela recepção dos dados respeitantes a um novo tipo de serviço ou edição de um já existente, e sua posterior inserção na base de dados.

- Funcionalidade – a página verifica se é uma inserção de um novo registo ou a edição de um existente.

  No primeiro caso, confirma se todos os dados necessários foram passados, sendo ainda verificado se a referência identificativa do operador é válida. Após isto instancia um novo objecto *ServiceType* com os dados recebidos, para fazer a inserção propriamente dita, passando esses dados para o *template* os mostrar.

  No segundo caso, verifica apenas que o identificador passado se refere a um *ServiceType* que exista registado, alterando em seguida os campos para os quais foram definidos valores. No final, passa esses valores para serem mostrados pelo *template*.
- Classes/módulos – é usada a classe *ServiceType* para inserção de um novo registo de tipo de serviço, para verificar a existência e alterar os valores de um determinado registo previamente existente. É usada a classe *Operator* para verificação de existência do operador referenciado pelos parâmetros passados. São ainda usados os módulos de *Session* e *Smarty*.
- Template – o *template* é "InsertServiceType.tpl", que essencialmente permite mostrar os dados que foram passados, para oferecer algum *feedback* visual ao administrador que os acabou de registar no sistema, sendo ainda mostrada uma mensagem a indicar o sucesso da operação (caso tenha existido).

AddIP.php – página que permite fazer a adição de uma nova gama de endereços IP, através da definição do endereço de subrede IP e a máscara de bits. Trata-se da realização do Caso de Utilização "Adicionar IP Range".

- Funcionalidade – a página é usada para fazer o preenchimento do formulário, bem como a própria inserção na base de dados, consoante os parâmetros passados contenham apenas o identificador do operador ou os dados completos para criar um novo registo de IP Range.
- Classes/módulos – é usada a classe Operator, para verificar se o identificador do operador é correcto e para fazer a inserção da nova gama de endereços. É igualmente usado o módulo Session, para controlo de sessões.
- Template – pode ser usado o *template* "AddIP.tpl", que apresenta um formulário para inserção dos valores de endereço de subrede e máscara de bits, sendo usado código Javascript para validação desses campos. O outro *template* que pode ser usado é "InsertIP.tpl", que exibe apenas uma mensagem a indicar se inserção teve sucesso ou não, mostrando neste caso o erro que foi detectado.

RemoveIP.php – página que permite concretizar a remoção de uma gama de endereços (correspondendo ao Caso de Utilização "Remover IP Range").

- Funcionalidade – verifica se foram passados os dados necessários (identificador de operador e endereço IP da subrede a remover) e se o identificador corresponde a um operador existente, procedendo em seguida à remoção da gama de endereços.
- Classes/módulos – é usada a classe *Operator*, para verificação da existência do operador e para a remoção. São ainda usados os módulos *Session* e *Smarty*.
- Template – o *template* usado para esta página é "RemoveIP.tpl". Permite mostrar uma mensagem informativa sobre o sucesso da operação, ou possíveis erros que tenham ocorrido. Exibe ainda uma ligação que permite voltar à página de edição do operador.

ListUsers.php – página que exibe a listagem de utilizadores registados no sistema, concretizando o Caso de Utilização "Listar Utilizadores". A listagem pode ser ordenada segundo vários critérios, podendo também ser definido o número de registo a mostrar de cada vez. É possível ainda proceder ao bloqueio/desbloqueio de utilizadores (correspondendo aos Casos de Utilização "Bloquear Utilizador" e "Desbloquear Utilizador").

- Funcionalidade – a página verifica se é um administrador que está a consultar, visto que apenas este tipo de utilizador pode aceder a ela. Em seguida, obtém os registos dos utilizadores que estão inseridos na base de dados. Em seguida realiza alguns cálculos para definir os registos a mostrar e eventualmente as ligações para a(s) página(s) anterior ou seguinte, visto que apenas é mostrada uma certa quantidade deles por página, conforme o que estiver definido na caixa de selecção. De referir ainda que os registos podem ser ordenados por *username*, *nome*, *data de registo* e *último login*, de forma ascendente ou descendente.
- Classes/módulos – a classe usada nesta página é a *NeteoUser*, para obter os registos de utilizadores. São ainda necessários os módulos *Session* e *Smarty*.
- Template – o *template* usado é "ListUsers.tpl", que contém uma secção *Smarty* para construir uma tabela com os registos atribuídos. No cabeçalho dessa tabela disponibiliza opções para ordenar os registos de forma ascendente/descendente, segundo o campo em causa. Para cada utilizador, tem a opção de o bloquear/desbloquear, consoante o seu estado seja "NORMAL" ou "BLOCKED".

  Consta ainda da página uma caixa de selecção que permite escolher o número de registos a mostrar de cada vez, bem como uma opção que alterna entre a exibição de todos os utilizadores ou apenas aqueles que se encontra activos (estado "NORMAL").

BlockUser.php – página responsável pelo bloqueio de um determinado utilizador.

- Funcionalidade – após confirmar que o identificador do utilizador a bloquear foi passado, verifica que esse utilizador existe e se ele não se encontra já bloqueado, caso em que procede ao seu bloqueio. Caso tenha havido algum erro, passa a mensagem correspondente ao *template*.
- Classes/módulos – Para ser possível a verificação da existência do utilizador, do seu estado e o bloqueio, é necessário usar a classe *NeteoUser*. Os módulos *Session* e *Smarty* são igualmente usados.
- Template – é usado o *template* "BlockUser.tpl", que apenas mostra uma mensagem indicando o estado da operação de bloqueio e uma ligação para retornar à página com a lista de utilizadores.

UnblockUser.php – página que permite efectuar o desbloqueio de um utilizador que se encontre bloqueado.

- Funcionalidade – são feitas verificações similares às utilizadas na página anterior, sendo que neste caso o estado do utilizador deve ser "NORMAL", para que o desbloqueio prossiga normalmente. Se tiver sido identificado algum erro, é passada uma mensagem de texto ao *template*.
- Classes/módulos - são usados os mesmos módulos e a mesma classe (*NeteoUser*), para verificação e alteração do estado para "NORMAL".
- Template – o *template* "UnblockUser.tpl" é similar ao *template* da página anterior.

InsertMirrorMeasurement.php – página responsável por receber os dados referentes às medições efectuadas pelos utilizadores, a partir do applet disponibilizado pelos mirrors, e por obter o valor do atraso RTT (ping). Permite ainda ao utilizador autenticado fazer a associação da medição feita a um dos contratos que tem registado no sistema.

- Funcionalidade – a partir dos parâmetros passados, a página distingue entre a inserção de uma medição e a associação de uma medição já registada a um tipo de contrato que o utilizador tem. Em ambos os casos confirma se todos os parâmetros necessários foram passados. No caso da associação, são feitas algumas verificações de segurança para evitar abusos por parte dos utilizadores (confirma se o utilizador e o seu endereço IP são iguais ao do registo da medição, se a medição não está já associada a um contrato e se o tipo de contrato indicado faz parte dos contratos registados do utilizador). Se tudo estiver correcto, efectua a associação ao contrato. Após isto, decide se a medição deve ser considerada confiável ou não, através da comparação entre os valores de download e upload da medição e os valores correspondentes do contrato associado. Caso os valores da medição excedam de forma significativa (mais de 50%) os do contrato, a medição é marcada como não sendo confiável e é reduzido o nível de confiança para o utilizador em questão. Se não exceder, a medição é declarada como confiável e é aumentado o nível de confiança que se tem no utilizador. Caso se trate da inserção de uma nova medição, são feitos cálculos para fazer a verificação do valor de controlo, é obtido o valor do atraso através de uma chamada ao sistema para execução do comando "ping", é pesquisado o operador ao qual pertence o endereço IP que o utilizador apresenta, é verificado se o mirror referenciado existe e é obtida a lista de contratos, no caso de se tratar de um utilizador autenticado que possua contratos registados. De referir ainda que os dados da medição são passados ao template, para que o utilizador os possa consultar, visto que é a obtenção desta informação que justifica o uso do applet.
- Classes/módulos – é usada a classe Measurement, para se realizarem as operações de inserção de medição, verificação e associação a contrato; é usada a classe Operator para pesquisar o operador através do IP do utilizador e disponibilizar o seu nome; é usada a classe NeteoUser para obter a sua lista de contratos, bem como para alteração do seu nível de confiança; é usada a classe Mirror para verificação da existência do mirror referenciado. De referir ainda o uso dos módulos Session e Smarty.
- Template – o template usado é "InsertMirrorMeasurement.tpl". Quando é inserida uma medição, apresenta os seus dados e a lista de contratos, no caso do utilizador se encontrar autenticado. Ao escolher um contrato da lista, inicia o mecanismo de associação (concretizando o Caso de Utilização "Associar medição a contrato"). É ainda disponibilizada uma opção que permite adicionar um novo contrato no momento, ficando a medição associada a ele no fim do processo.
  Quando se trata da associação, é apenas apresentada uma mensagem a indicar como correu a operação.
  Na ocorrência de algum erro, é igualmente apresentada apenas uma mensagem a informar o utilizador do sucedido.

ListNews.php – página que apresenta uma listagem com as notícias registadas no sistema, realizando assim o Caso de Utilização "Listar Notícias", mostrando para cada uma a sua data de inserção e o seu título, bem como algumas opções.

- Funcionalidade – após verificar que o utilizador que está a consultar a página é um administrador, obtém os registos das notícias, podendo a estes ser aplicado um limite de quantidade e um *offset*, se isso estiver definido nos parâmetros passados e se houverem notícias inseridas na base de dados. Após isto faz a atribuição a passagem desses registos ao *template*.
- Classes/módulos – é utilizada a classe *News* para verificação do número de registos de notícias existentes bem como para se ter acesso aos dados desses registos.
- Template – o *template* "ListNews.tpl" apresenta uma tabela onde são mostrados dados relativos às notícias listadas (data de inserção e título), bem como opções para visualizar, editar ou remover relativamente a cada uma delas (permitindo assim iniciar os Casos de Utilização "Consultar Notícia", "Editar Notícia" e "Remover Notícia"). Disponibiliza ainda uma opção para adicionar uma nova notícia (correspondendo ao Caso de Utilização "Adicionar Notícia").

EditNews.php – página que permite criar uma nova notícia, consultar ou editar os dados de uma notícia já existente (permitindo a realização dos Casos de Utilização "Adicionar Notícia", "Consultar Notícia" ou "Editar Notícia").

- Funcionalidade – conforme os parâmetros passados, a página decide se se trata de uma nova inserção, consulta ou edição, escolhendo o *template* a usar. No caso destas duas últimas situações, é ainda pesquisada a notícia em causa e os seus dados são passados ao *template*.
- Classes/módulos – para obtenção dos dados da notícia, é necessário o recurso à classe News.
- Template – são usados dois *templates*: "ViewNews.tpl" e "EditNews.tpl".

  No primeiro, são apresentados os dados completos da notícia – data, título e corpo – assim como opções para a sua edição ou remoção.

  No segundo, é disponibilizado um formulário, que se encontra preenchido com os dados da notícia no caso de ser uma edição. Os campos de preenchimento obrigatório são verificados com o auxílio de código Javascript.

  A submissão dos dados é feita para a página "InsertNews.php".

InsertNews.php – está página é responsável pela verificação e inserção dos dados provenientes do formulário da página "EditNews.php".

- Funcionalidade – é feita a distinção entre a edição de uma notícia existente e a inserção de uma nova notícia. No primeiro caso, é instanciada uma nova *News* a partir da data passada em parâmetro, sendo em seguida alterados os campos necessários.

  No caso da inserção, é igualmente instanciada uma nova *News*, usando o título e corpo que são passados.

  Em ambos os casos, os valores dos campos da notícia são passados ao *template*.
- Classes/módulos – recorre-se ao uso da classe *News* para realizar as operações atrás descritas, com vista a inserir novo registo ou alterar os campos de um registo existente.
- Template – o *template* "InsertNews.tpl" mostra ao utilizador os campos da notícia, bem como uma mensagem a indicar o sucesso (ou insucesso) da operação. São ainda disponibilizadas opções para voltar à página principal ou para ir para a página que faz a listagem das notícias.

SACheckAuth.php – esta página permite à aplicação stand-alone verificar a correcção de um par utilizador/palavra-chave. Tal é usado na interface de opções.

- Funcionalidade – Simplesmente retorna "Good" ou "Bad" conforme os dados enviados sejam ou não válidos.
- Classes/módulos – recorre apenas ao uso do módulo de sessão, e desta forma, indirectamente, à classe de utilizador.
- Template – Este é dos poucos ficheiros que não possui qualquer template já que apenas imprime uma simples String.

SAGetMessages.php – esta página permite à aplicação stand-alone obter mensagens de alerta que poderão ser mostradas ao iniciar a aplicação.

- Funcionalidade – Conforme a versão indicada pela aplicação, pode ou não devolver uma mensagem de alerta, por exemplo, indicando que deve actualizar a aplicação. As mensagens e as condições são definidas directamente no código.
- Classes/módulos – Não recorre a qualquer classe ou módulo.
- Template – Este é dos poucos ficheiros que não possui qualquer template já que apenas imprime uma simples String.

SAGetState.php – esta página permite à aplicação obter o estado do tempo para o utilizador actual, de modo a saber qual o ícone a ser representado na bandeja do sistema.

- Funcionalidade – Faz uma consulta das últimas medições do utilizador, das percentagens médias e devolve um estado segundo critério idêntico ao utilizado na geração do mapa de entrada do site.
- Classes/módulos – Foi necessário, antes de mais, de utilizar o módulo de controlo de sessão. Para além disso, para obter as informações das medições, a classe measurement. Para além disto, as quase sempre obrigatórias classes de templates e de utilizador.
- Template – É utilizado o template "SAGetState.tpl" em grande parte para facilitar a adaptação do código usado no gerador de mapas. Apenas contém a indicação de uma string.

SAInsertMirrorMeasurement.php – esta página é a que recebe os resultados de uma medição efectuada pelo Stand-Alone.

- Funcionalidade – As funcionalidades desta página são em quase tudo idênticas à do InsertMirrorMeasurement.php, a "irmã" para a recepção de dados de medições *on-line*. As principais diferenças prendem-se com o facto de aqui ser realizada sempre a associação imediatamente e ao facto de não haver um controlo de alteração de endereço, já que este não pode ser editado como no web browser.
- Classes/módulos – Os módulos/classes base estão presentes aqui. São eles o de sessão, o de templates e o do utilizador. Para além deste recorre-se ainda à classe de gestão de medições para se proceder á sua inserção.
- Template – É utilizado o template "SAInsertMirrorMeasurement.tpl" em parte para facilitar a adaptação do código do InserMirrorMeasurement. Mostra uma lista de strings de erros caso eles existam, separadas pelo carácter especial '£'. É indicado ainda o

resultado do teste de ping para que possa ser visualizado no interface de testes explícitos da aplicação, separando este valor com o carácter especial '§'.

SAListContracts.php – esta página permite ao Stand-Alone consultar a lista de contractos que podem ser seleccionados na interface principal

- Funcionalidade – Lista os tipos de serviço que o utilizador tem registado, filtrando aqueles que são as alternativas viáveis a partir do operador detectado através do endereço IP da máquina.
- Classes/módulos – Os módulos/classes base estão presentes aqui. São eles o de sessão, o de templates e o do utilizador. Para efectuar a detecção do operador é ainda necessário o uso da classe associada a este.
- Template – É utilizado o template "SAListContracts.tpl". Apesar da ausência de design visual, torna-se útil na impressão da lista dos contratos. Mostra uma lista de contratos identificados por um trio de nome, localidade e identificador. Para separar contratos usaram-se sequências específicas de caracteres, improváveis de ocorrer dentro do nome dos tipos de contrato.

Anexo D – Casos de Utilização Reais

Na presente secção são descritos os diálogos através dos quais os utilizadores podem interagir com o sistema. Cada descrição exibe tipicamente um ou mais *screenshots/layouts* que reflectem a forma como os casos de utilização foram concretizados na prática, sendo de igual modo apresentada a narrativa detalhada, descrevendo de forma pormenorizada os passos que envolvem a sua realização habitual bem como sequências alternativas, com referências para os *layouts*.

Esta abordagem tem o intuito de mostrar a forma como os casos de utilização foram "passados à prática", ajudando no conhecimento do sistema que foi criado e no modo como deve ser usado para se usufruir das funcionalidades pretendidas.

Efectuar registo

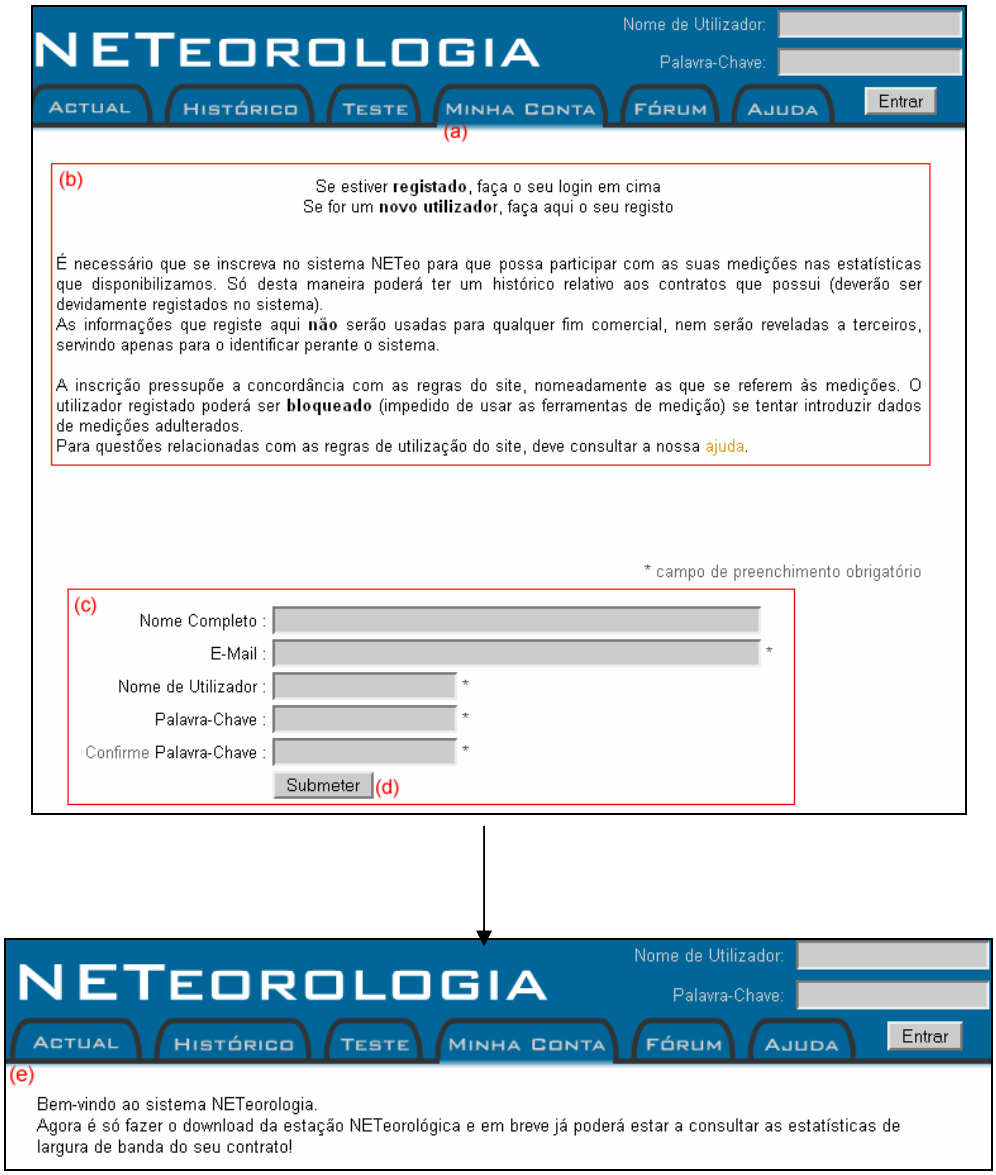


Figura 25 – *Layout* Efectuar Registo

Nome:	**Efectuar Registo**	
Âmbito:	NETeorologia	
Finalidade:	Criação de uma nova conta no sistema, fornecendo informação pessoal, sendo esta identificada por um nome de utilizador e palavra-chave.	
Actores:	Utilizador anónimo	
Tipo:	Real	
Descrição geral:	Actor decide efectuar um novo registo no sistema. Sistema mostra formulário que utilizador preenche. Submete e é validado o registo se dados forem correctos.	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Clica em (a) para efectuar o registo.	
		2. O sistema mostra ao actor uma nova página com condições gerais para o registo no site em (b). Apresenta igualmente um formulário com campos de preenchimento em (c): nome, endereço de correio electrónico, nome de utilizador, palavra-chave (campo duplo para confirmação).
	3. O utilizador preenche campos de formulário em (c)	
	4. Submete dados clicando em (d).	
		4. Sistema verifica se todos os campos estão preenchidos correctamente.
		5. Mostra uma nova página indicando o sucesso da operação em (e). CaU termina, após ter feito o *login* deste novo utilizador.
Sequências alternativas e extensões:		A1. Se o sistema verificar que ocorreram erros em 4, volta a 2, sinalizando os campos onde houve erros.

Efectuar *login*

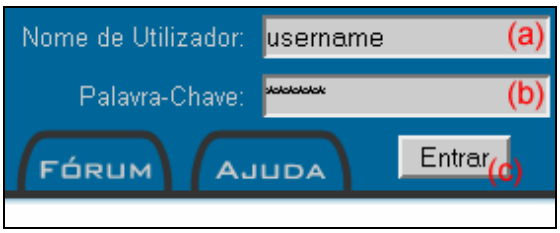


Figura 26 – *Layout* Efectuar *Login*

Nome:	**Efectuar *Login***	
Âmbito:	NETeorologia	
Finalidade:	O actor identifica-se perante o sistema através do seu nome de utilizador e palavra-chave, de forma a poder usufruir dos serviços disponíveis para utilizadores registados.	
Actores:	Utilizador anónimo	
Tipo:	Real	
Descrição geral:	Utilizador insere nome de utilizador e palavra-passe para identificação por parte do sistema. Sistema faz *login* do utilizador conforme os dados estejam correctos ou não.	
Sequência típica dos eventos:	**Actor** 1. Insere nome de utilizador em (a) e palavra-chave em (b), clicando em seguida em (c) para submeter dados ao sistema.	**Sistema** 2. Sistema verifica que dados estão correctos e inicia nova sessão para este utilizador. Termina CaU.
Sequências alternativas e extensões:		A1. Caso dados sejam incorrectos, mostra mensagem de erro.

Efectuar *logout*

Figura 27 – *Layout* Efectuar *logout*

Nome:	**Efectuar *Logout***	
Âmbito:	NETeorologia	
Finalidade:	O actor indica ao sistema que pretende terminar a sessão como utilizador registado.	
Actores:	Utilizador registado	
Tipo:	Real	
Descrição geral:	Utilizador clica na opção que lhe permite terminar a sessão.	
Sequência típica dos eventos:	**Actor** 1. Clica em (a)	**Sistema** 2. Sistema termina sessão do utilizador e redirecciona para página inicial. Termina CaU
Sequências alternativas e extensões:	(Não tem)	

Realizar medição *on-line*

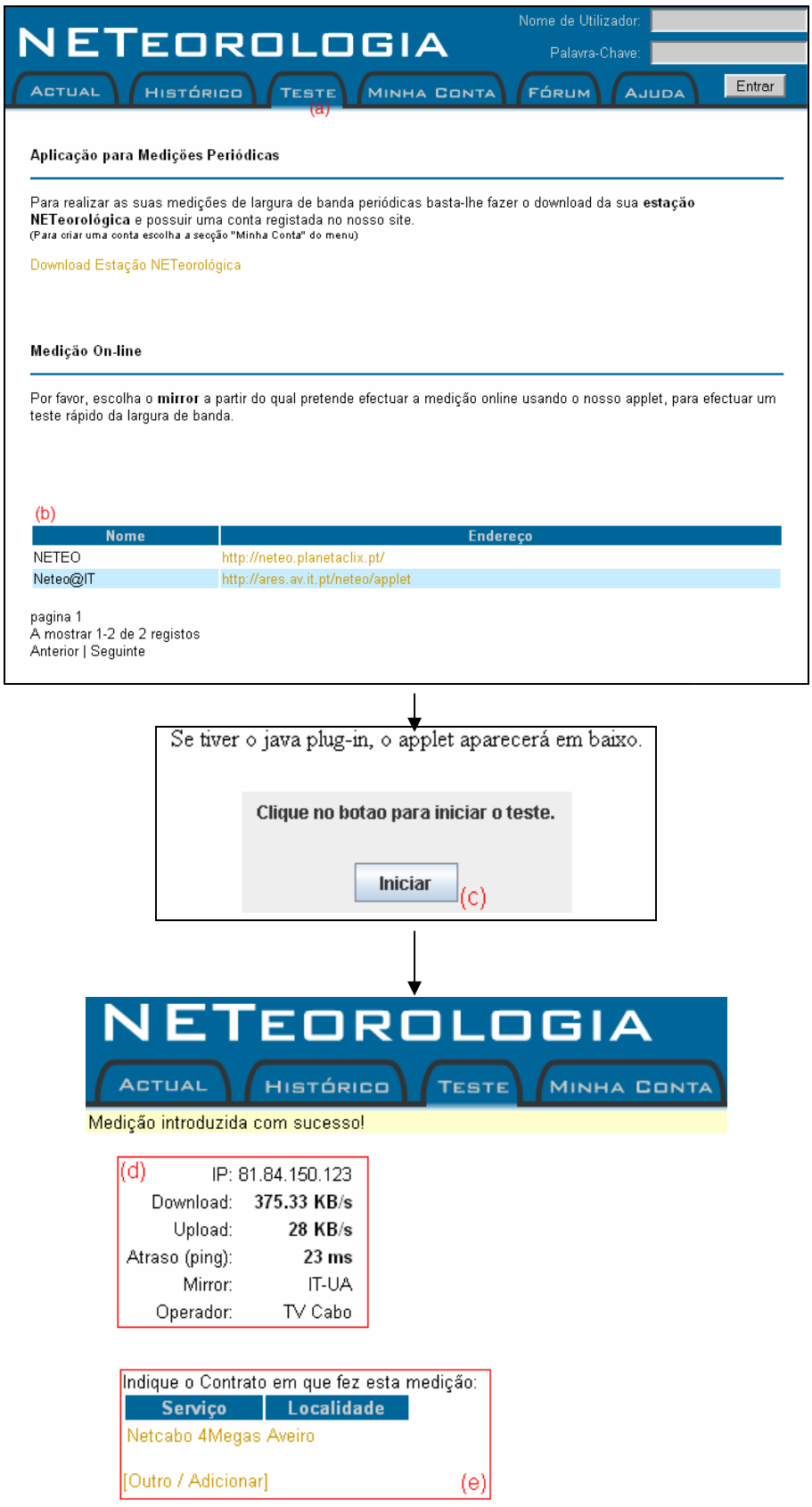


Figura 28 – *Layout* de Realizar medição *on-line*

Nome:	**Realizar medição *on-line***	
Âmbito:	NETeorologia	
Finalidade:	O utilizador anónimo ou o utilizador registado utilizam a ferramenta *on-line* para efectuar uma medição pontual da largura de banda da ligação do utente.	
Actores:	Utilizador registado, utilizador anónimo	
Tipo:	Real	
Descrição geral:	Utilizador selecciona a página para fazer teste, escolhe *mirror* que pretende e realiza medição. Sistema mostra resultados da medição.	
Sequência típica dos eventos:	**Actor**	**Sistema**
	1. Utilizador clica em (a), indo para a página de "Teste".	
		2. Mostra os servidores (*mirrors*) que estão disponíveis para serem usados para o teste, em (b).
	3. Escolhe o servidor que pretende usar.	
		4. Inicia o carregamento do *applet* de medição a partir do servidor escolhido
	5. Clica em (c), para iniciar teste.	
		6. *Applet* faz os testes de *download*/*upload* de ficheiros, mostrando barra com estado de evolução do teste.
		7. Após fim do teste, mostra resultados, em (d).
		8. Armazena resultados no sistema. CaU termina.
Sequências alternativas e extensões:		A1. Caso utilizador esteja autenticado, apresenta lista de contratos registada para o utilizador, em (e), podendo este iniciar assim o CaU "Associar Medição a Contrato". São listados os contratos referentes ao operador identificado pelo endereço IP do utilizador e que possam atingir os valores de *download* da medição.

Associar Medição a Contrato

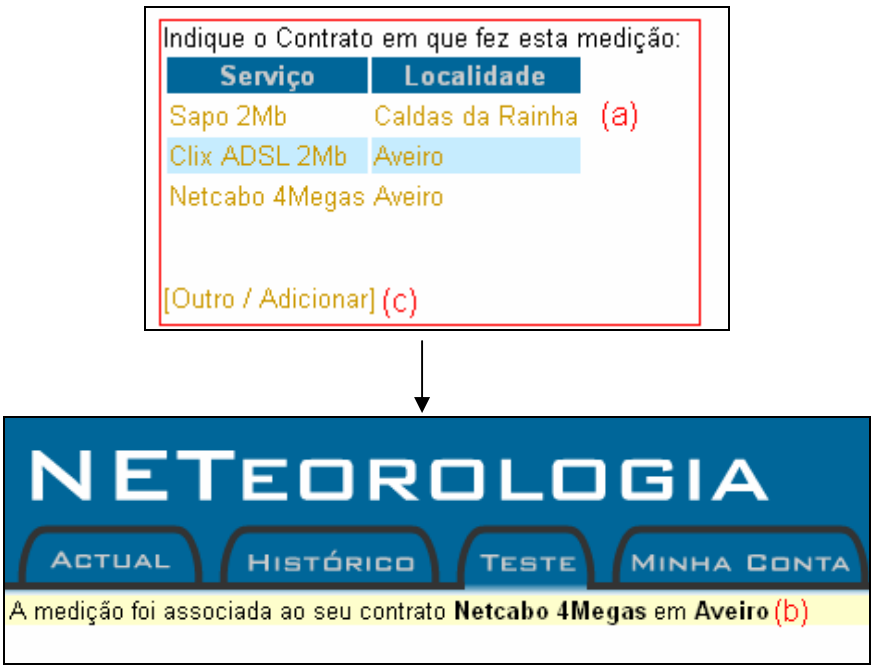


Figura 29 – *Layout* Associar Medição a Contrato

Nome:	**Associar medição a contrato**	
Âmbito:	NETeorologia	
Finalidade:	É feita uma associação entre a medição feita pelo utilizador registado e um dos contratos que possui na sua conta.	
Actores:	Utilizador registado	
Tipo:	Real	
Descrição geral	Ao ser inserida uma medição, sistema mostra lista de contratos do actor. Este escolhe o contrato ao qual pretende associar a medição feita. Sistema regista associação.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Mostra contratos que constam da conta do utilizador.
	2. Escolhe contrato ao qual pretende associar esta medição, em (a).	
		3. Associa medição ao contrato indicado pelo actor. Mostra mensagem em (b) e termina.
Sequências alternativas e extensões:	A1. Caso pretenda, pode adicionar outro contrato à sua conta, escolhendo (c). Após completar o CaU correspondente de "Adicionar Contrato", a medição fica associada a este novo contrato.	

Visualizar estatística

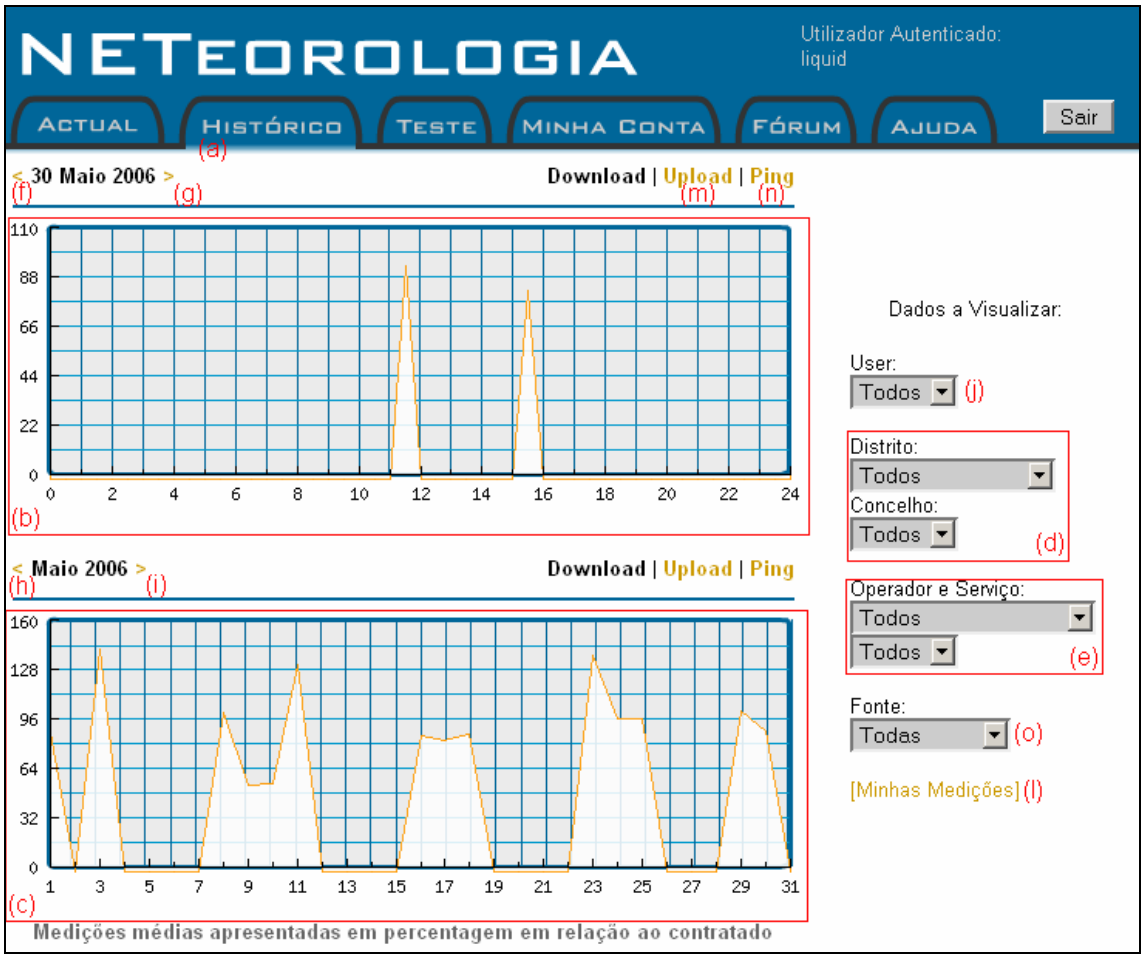


Figura 30 – *Layout* Visualizar Estatística

Nome:	**Visualizar estatística**	
Âmbito:	NETeorologia	
Finalidade:	O sistema mostra informação tratada estatisticamente de forma gráfica referente às medições, podendo o utilizador escolher entre vários parâmetros de filtragem.	
Actores:	Utilizador registado, Utilizador anónimo	
Tipo:	Real	
Descrição geral:	Utilizador selecciona página de estatísticas e visualiza a informação, podendo escolher diversos parâmetros de filtragem.	
Sequência típica dos eventos:	**Actor** 1. Actor selecciona (a) 3. Escolhe operador e contrato em (d), local (distrito e concelho) em (e), sobre o qual quer ver estatísticas.	**Sistema** 2. Sistema apresenta página de gráficos de medições globais em (b) e (c), correspondendo a dia e mês, com valores relativos (percentual em relação ao contrato declarado).

Nome:	**Visualizar estatística**	
	5. Actor consulta informação. Termina CaU.	4. Mostra gráficos relativos às estatísticas do dia e mês actuais para as escolhas pelo actor.
Sequências alternativas e extensões:	A1. Actor pode consultar dados de dia seguinte, clicando em (g), ou do dia anterior, clicando em (f). Pode fazer igual operação em relação ao mês, escolhendo (i) para avançar ou (h) para retroceder. B1. Se pretender pode visualizar apenas as suas próprias estatísticas, clicando em (j). nota: esta opção apenas está disponível para utilizador registado. C1. Ao escolher (l), inicia o caso de utilização "Visualizar minhas medições". D1. Se pretender visualizar as estatísticas de *upload* ou de *ping* (atraso RTT), deve escolher (m) ou (n) respectivamente. E1. Caso queira ver as medições por tipo de fonte (*web applet* ou aplicação *standalone*), deve clicar em (o).	F1. Em 4, as estatísticas são apresentadas em kilobits/s, caso seja *download* ou *upload*, ou em milissegundos caso se trate de *ping.*

Visualizar medições

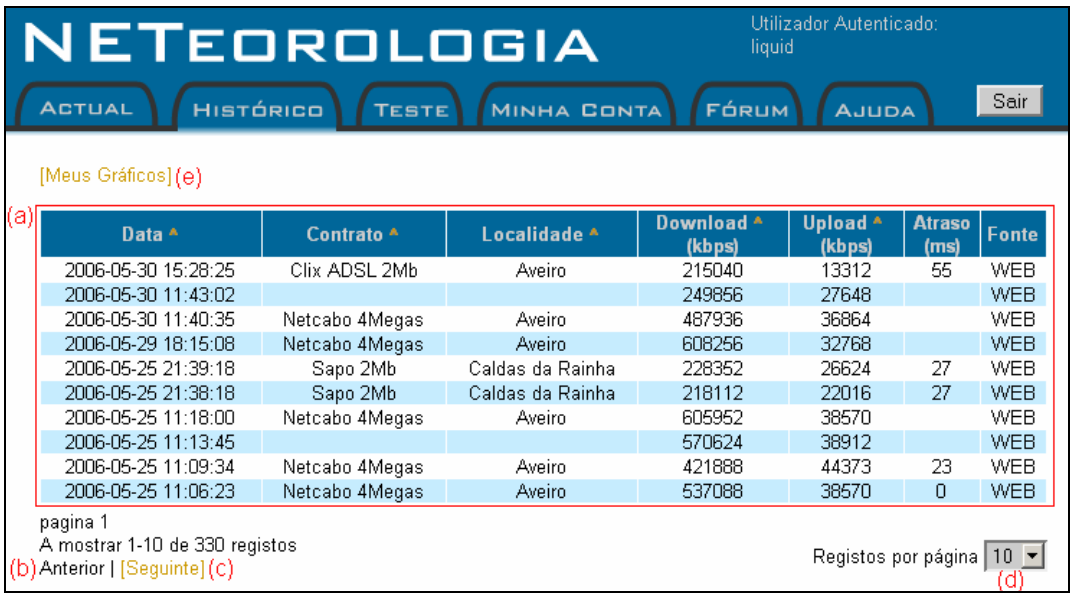


Figura 31 – *Layout* Visualizar Medições

Nome:	**Visualizar medições**	
Âmbito:	NETeorologia	
Finalidade:	O sistema mostra lista com as últimas medições efectuadas pelo utilizador	
Actores:	Utilizador registado	
Tipo:	Real	
Descrição geral:	Actor selecciona página com as suas medições. Sistema apresenta lista das medições, caso existam. Actor pode ordenar a lista segundo vários parâmetros.	
Sequência típica dos eventos:	**Actor** 2. Actor consulta informação. Termina CaU.	**Sistema** 1. Sistema apresenta listagem dos registos das medições feitas pelo utilizador em (a), mostrando os seus dados (data, tipo de contrato e sua localização, valor de *download*, *upload* e atraso, bem como a fonte).
Sequências alternativas e extensões:	A1. Se pretender, pode ordenar a lista de forma ascendente ou descendente, clicando nos cabeçalhos que possuem (b). B1. Caso tenha mais registos antes ou depois desta página, pode avançar em (d) ou retroceder em (c) para eles. O número de registos pode ser escolhido em (e). C1. Ao escolher (f), visualiza gráficos das suas medições.	D1. Caso o utilizador não possua medições, o sistema sinaliza o facto ao utilizador. E1. Se utilizador não estiver autenticado, sinaliza erro.

Consultar dados de conta

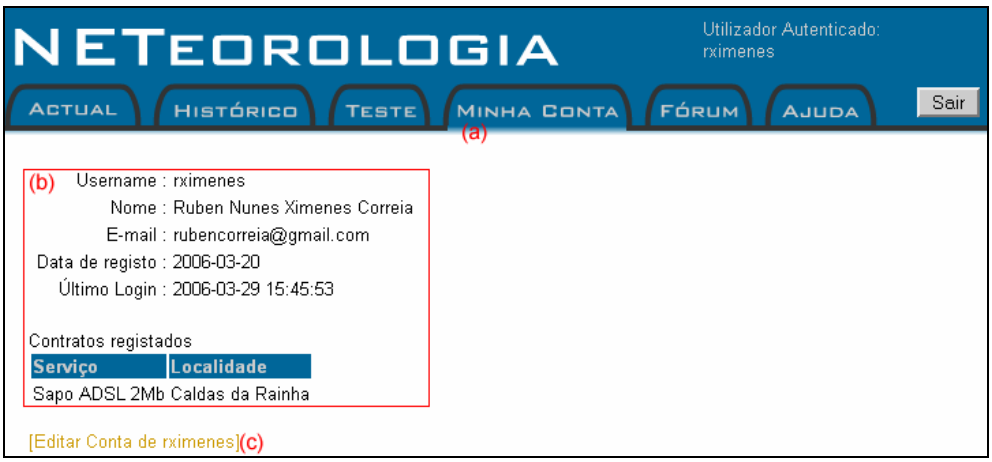


Figura 32 – *Layout* Consultar dados de conta

Nome:	**Consultar dados de conta**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador registado faz a consulta dos dados que constam na sua conta.	
Actores:	Utilizador registado	
Tipo	Real	
Descrição geral	Actor escolhe opção que lhe permite visualizar os dados da sua conta.	
Sequência típica dos eventos:	**Actor** 1. Actor clica em (a) 3. Actor consulta dados. CaU termina.	**Sistema** 2. Sistema mostra página onde apresenta dados de conta do utilizador, em (b)
Sequências alternativas e extensões:	A1. Se actor quiser editar a conta, pode escolher (c) na interface, iniciando o CaU "Editar dados de conta".	

Editar dados de conta

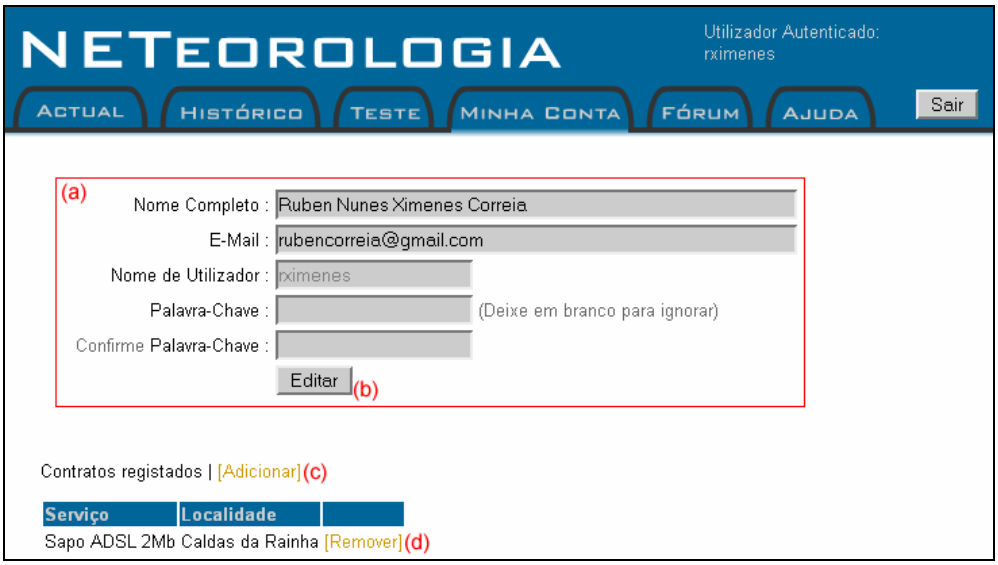


Figura 33 – *Layout* Editar dados de conta

Nome:	**Editar dados de conta**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador registado faz a alteração dos dados que constam na sua conta e insere essas alterações no sistema.	
Actores:	Utilizador registado, Utilizador anónimo	
Tipo:	Real	
Descrição geral:	Utilizador altera dados da sua conta. Sistema guarda alterações se estiverem correctas.	
Sequência típica dos eventos:	**Actor** 2. Actor altera dados que pretende em (a) 3. Submete dados clicando em (b)	**Sistema** 1. Sistema mostra (a), preenchido com dados de conta já registados. 4. Sistema verifica consistência dos dados submetidos. 5. Mostra nova página com mensagem de sucesso da operação. CaU termina.
Sequências alternativas e extensões:	B1. Pode pretender adicionar novo contrato, ao clicar em (c). C1. Pode igualmente remover um contrato, clicando em (d).	A1. Em 5, sistema pode encontrar erros nos dados submetidos. Volta a 2, indicando em (b) onde ocorreram erros.

Adicionar contrato

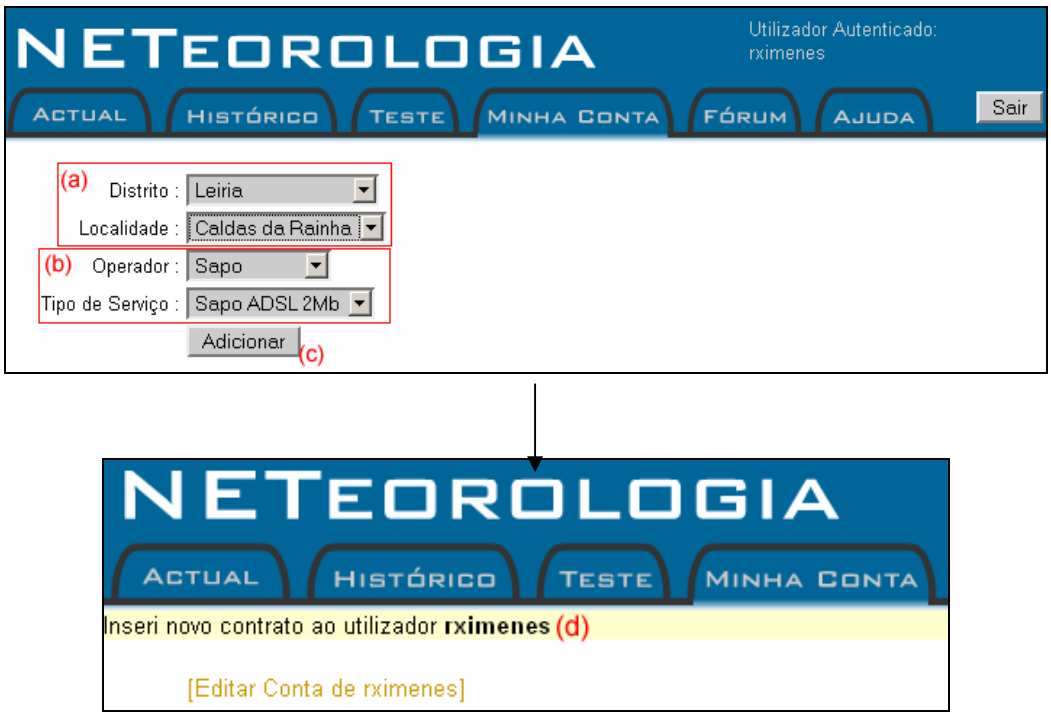


Figura 34 – *Layout* Adicionar contrato

Nome:	**Adicionar Contrato**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador registado indica ao sistema que pretende associar um novo contrato ao seu registo.	
Actores:	Utilizador registado	
Tipo:	Real	
Descrição geral:	Utilizador indica ao sistema a adição de um novo contrato à sua conta. Sistema guarda alterações.	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe a opção adequada na interface. 3. Escolhe localização em (a) e o contrato que pretende em (b). 4. Submete escolhas em (c).	**Sistema** 2. Sistema mostra nova página com campos para escolha. 5. Mostra sucesso da operação em (d). CaU termina.
Sequências alternativas e extensões:		A1. Caso tenha havido algum problema, avisa o utilizador do facto em (d).

Desactivar contrato

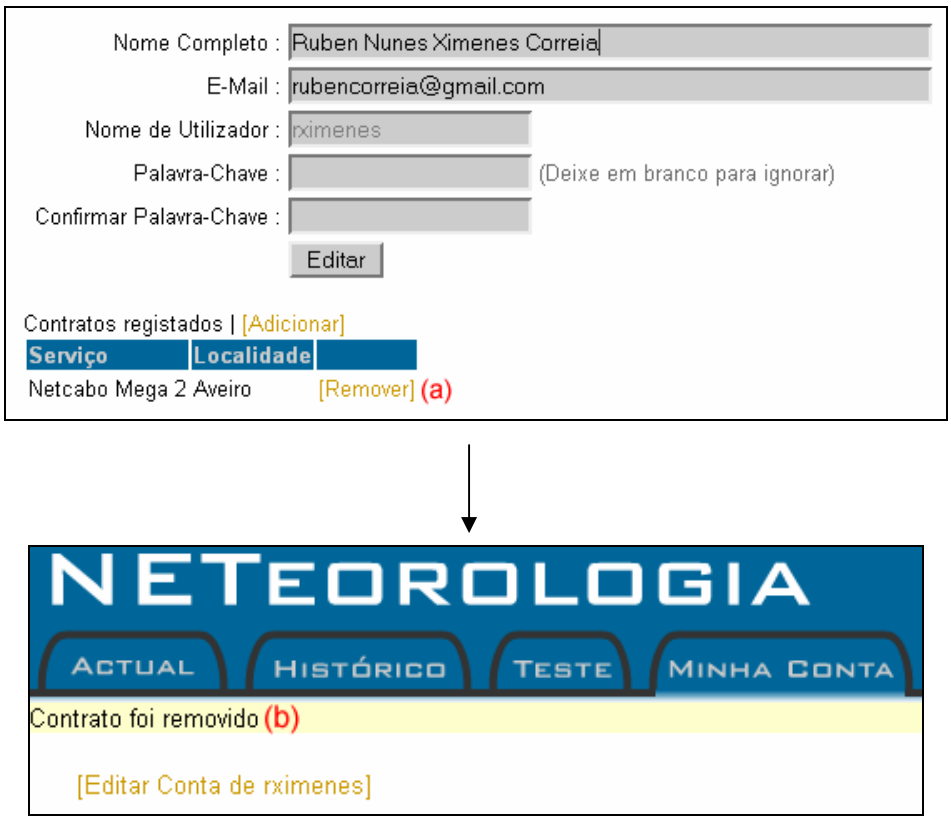


Figura 35 – *Layout* Remover contrato

Nome:	**Desactivar Contrato**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador registado indica ao sistema que pretende desactivar da sua conta um contrato.	
Actores:	Utilizador registado	
Tipo:	Real	
Descrição geral:	Utilizador indica ao sistema a desactivação de um contrato que tinha para a sua conta. Sistema guarda alterações.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Sistema mostra lista de contratos registados pelo utilizador.
	2. Actor selecciona em (a) o contrato a desactivar.	
		5. Sistema sinaliza na conta do utilizador a que o contrato se encontra desactivado.
		6. Apresenta mensagem de sucesso da operação em (b). CaU termina.
Sequências alternativas e extensões:	(Não tem)	

Configurar aplicação

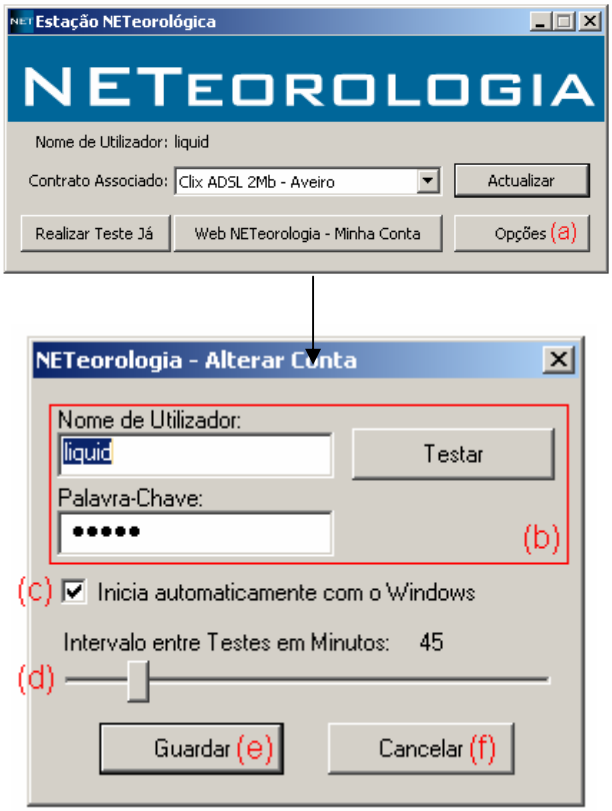


Figura 36 – *Layout* Configurar aplicação

Nome:	**Configurar aplicação**	
Âmbito:	NETeorologia	
Finalidade:	O utilizador configura as opções da aplicação.	
Actores:	Utilizador registado, utilizador anónimo	
Tipo:	Real	
Descrição geral:	Actor altera as opções de configuração da aplicação residente.	
Sequência típica dos eventos:	**Actor** 1. Escolhe (a) para editar opções. 2. Altera o que pretende: dados da conta em (b), podendo "Testar" para verificar a validade; início automático em (c); periodicidade em (d). 3. Clica em (e) para guardar alterações.	**Sistema** 1. Sistema mostra novo diálogo relativo à configuração. 4. Assume novas configurações. CaU termina.
Sequências alternativas e extensões:	A1. Pode cancelar ao escolher (f).	

Consultar Ajuda

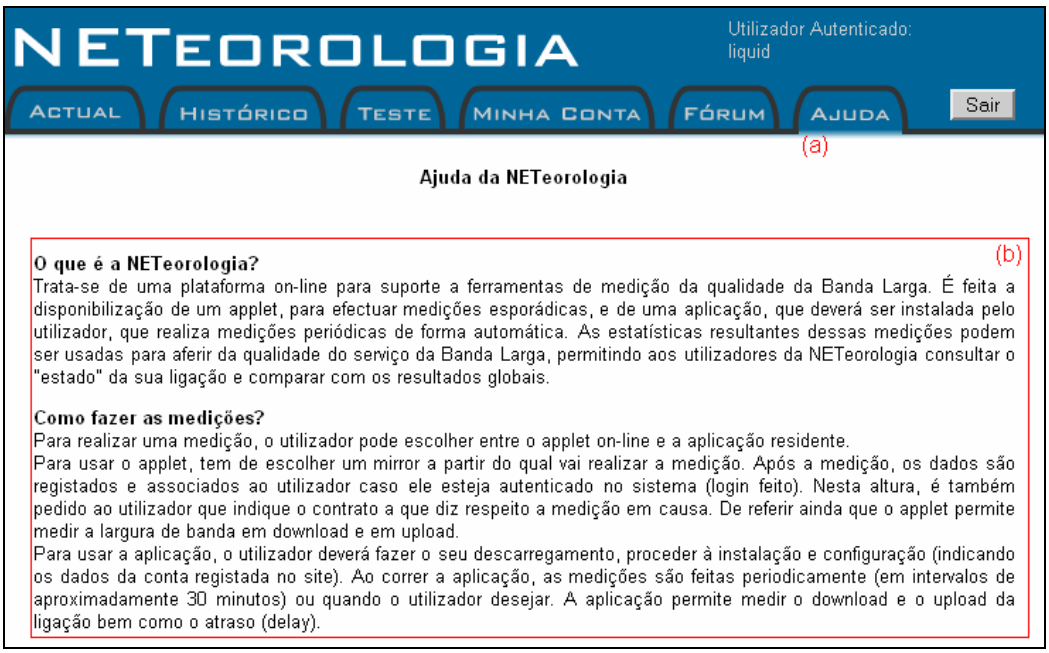


Figura 37 – *Layout* Consultar Ajuda

Nome:	**Consultar Ajuda**	
Âmbito:	NETeorologia	
Finalidade:	Utilizador consulta ajuda.	
Actores:	Utilizador anónimo	
Tipo:	Real	
Descrição geral:	Sistema mostra página de Ajuda ao actor	
Sequência típica dos eventos:	**Actor** 1. Escolhe (a). 2. Consulta Ajuda. CaU termina.	**Sistema** 2. Mostra página de Ajuda, em formato de FAQ. CaU termina.
Sequências alternativas e extensões:		

Listar operadores

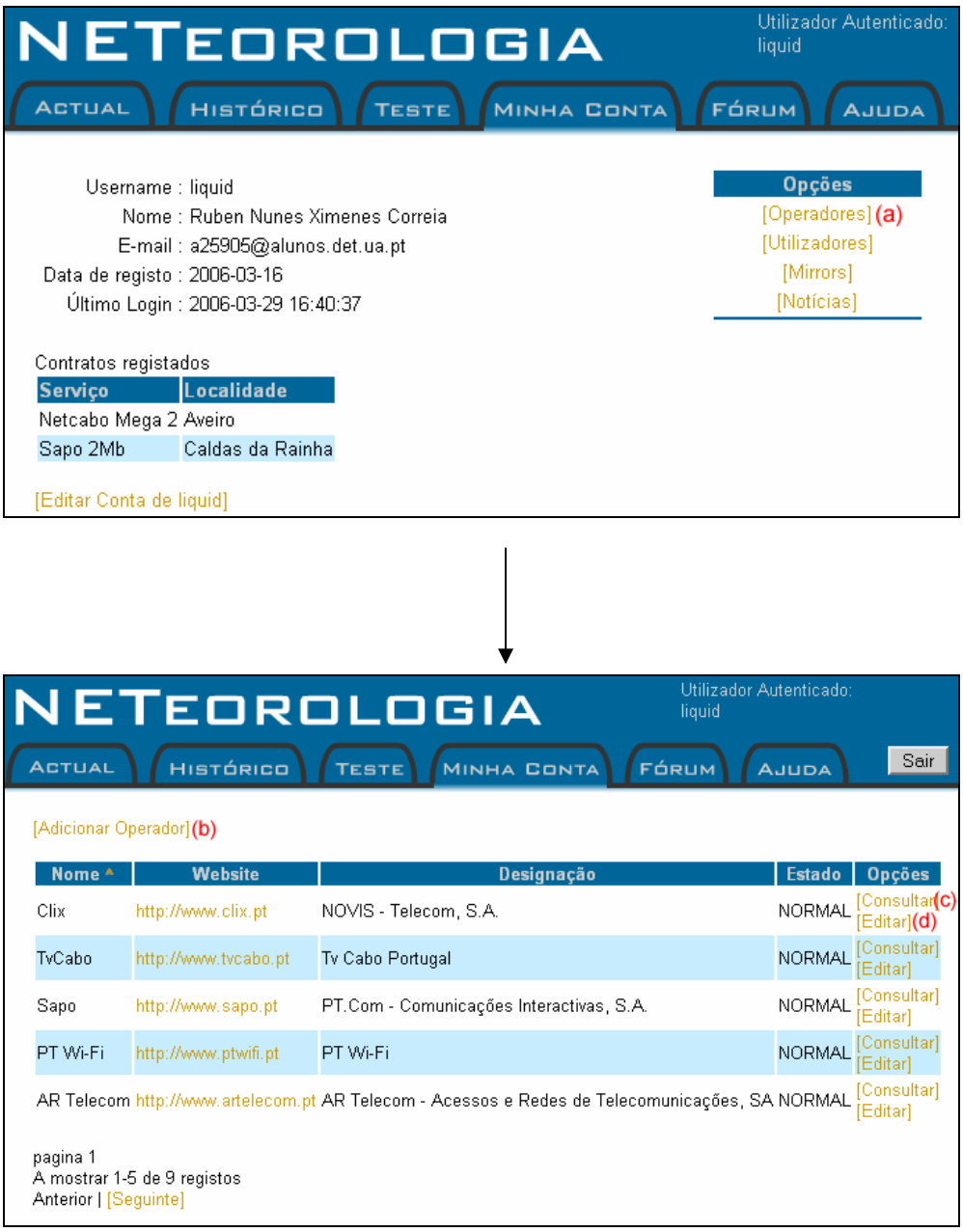


Figura 38 – *Layout* Listar Operadores

Nome:	**Listar operadores**	
Âmbito:	NETeorologia	
Finalidade:	Sistema mostra lista de operadores actualmente registados.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Administrador pede ao sistema para que lhe mostre a lista de operadores registados. Sistema devolve lista.	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe (a). 3. Actor consulta. CaU termina.	**Sistema** 2. Sistema mostra ecrã com lista dos operadores que estão registados no sistema, mostrando alguns dados (nome, *website*, designação, estado).
Sequências alternativas e extensões:	A1. Actor pode escolher (c) ou (d) para cada operador, podendo assim iniciar os CaU "Consultar dados de operador" e "Editar dados de operador". B1. Pode igualmente escolher (b), que inicia o CaU "Adicionar operador", para o operador em questão.	

Adicionar operador

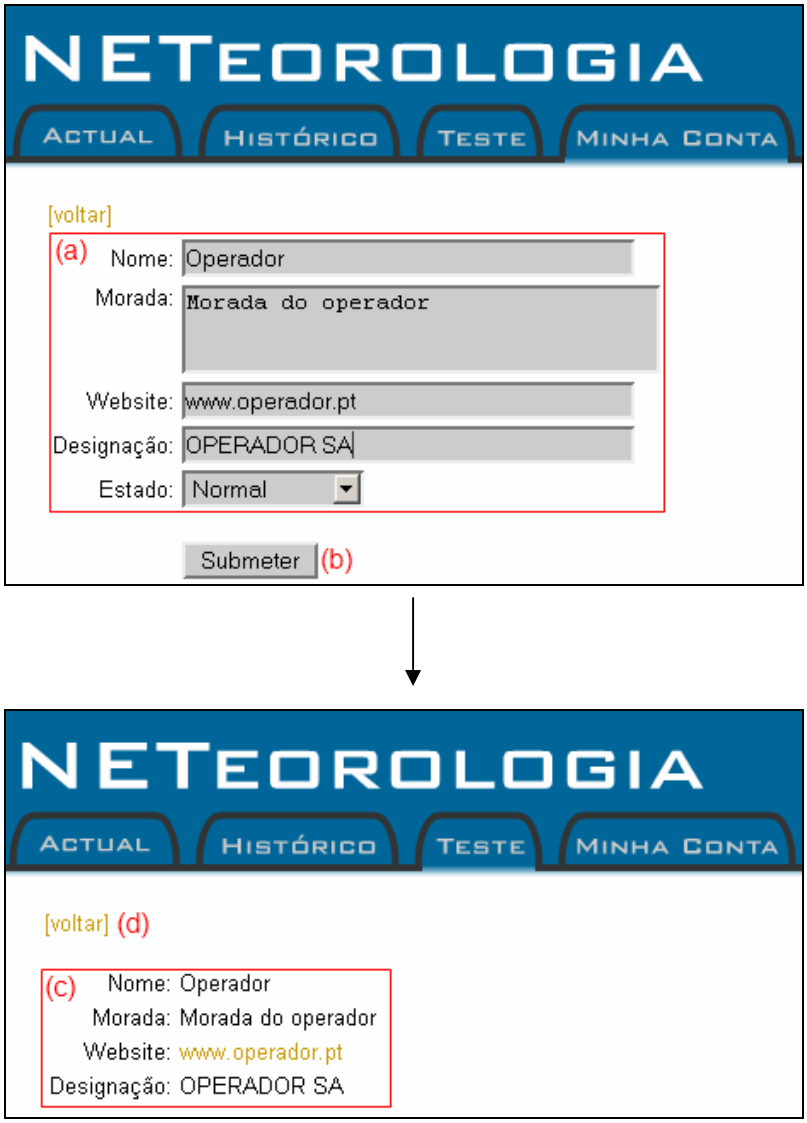


Figura 39 – *Layout* Adicionar Operador

Nome:	**Adicionar operador**	
Âmbito:	NETeorologia	
Finalidade:	O administrador adiciona ao sistema informação referente a um novo operador.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Administrador indica dados de novo operador. Sistema guarda registo desses dados.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Sistema mostra ecrã para inserção de dados relativos ao novo operador: nome, morada, sítio web, designação em (a). Todos os campos são de preenchimento obrigatório.
	2. Insere dados referentes ao novo operador.	
	3. Submete dados, clicando em (b).	
		4. Verifica consistência dos dados.
		5. Regista novo operador na base de dados.
		6. Mostra mensagem de sucesso da operação em (c). CaU termina.
Sequências alternativas e extensões:		A1. A verificação encontra erros nos dados que foram submetidos. Volta ao passo 1, indicando erros que ocorreram.
	B1. Ao clicar em (d), volta à lista de operadores.	

Consultar dados de operador

Figura 40 – *Layout* Consultar dados de operador

Nome:	**Consultar dados de operador**	
Âmbito:	NETeorologia	
Finalidade:	Sistema mostra ao administrador dados relativos a determinado operador	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Administrador selecciona operador e sistema mostra dados desse operador.	
Sequência típica dos eventos:	**Actor** 2. Actor consulta informação. CaU termina.	**Sistema** 1. Sistema mostra em (a) dados referentes ao operador seleccionado (designação, morada, sítio web, *IP-Ranges*, dados dos tipos de contrato que lhe estão associados).
Sequências alternativas e extensões:	A1. Ao consultar informação, actor pode pretender alterar dados, escolhendo (b) iniciando assim o CaU “Editar dados de operador”. B1. Pode igualmente clicar no nome de tipo de contrato, em (c), que lhe permite iniciar o CaU “Editar tipo de contrato”.	

Editar dados de operador

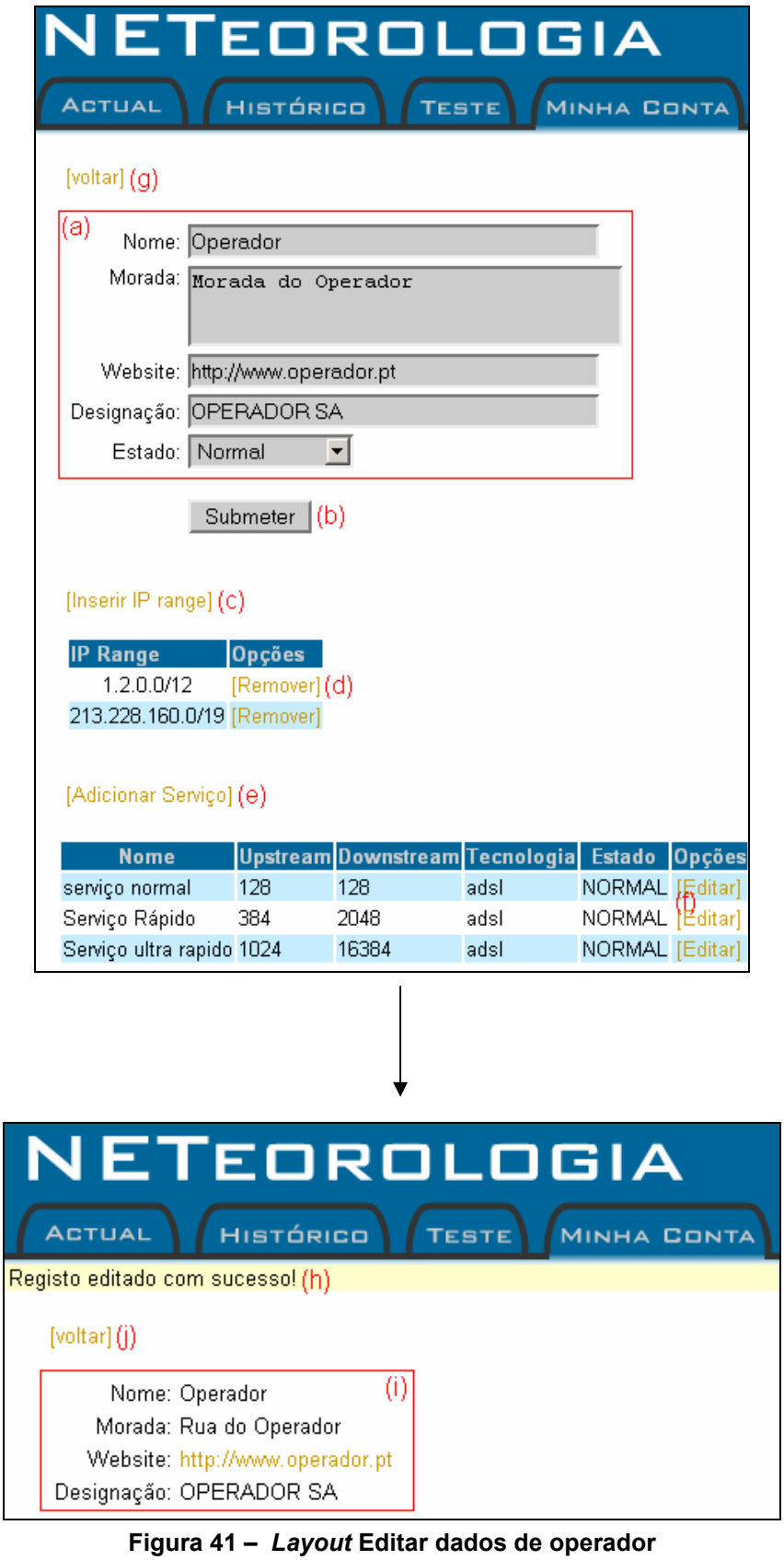


Figura 41 – *Layout* Editar dados de operador

Nome:	**Editar dados de operador**	
Âmbito:	NETeorologia	
Finalidade:	Administrador altera os dados relativos a determinado operador	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Após seleccionar operador, administrador altera dados do mesmo e submete-os. Sistema verifica consistência de dados e regista alterações.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Sistema mostra formulário em (a), com campos preenchidos pelos dados actualmente constantes no sistema.
	2. Actor muda dados que pretende.	
	3. Submete alterações em (b).	
		4. Verifica consistência de dados.
		5. Regista alterações no sistema.
		6. Mostra mensagem de sucesso da operação, em (h), em conjunto com os dados do operador em (i).
Sequências alternativas e extensões:	A1. Se pretender adicionar um IP range, escolhe (c). Para remover um IP range já existente, clica em (d). B1. Ao escolher (e), pode iniciar o CaU "Adicionar tipo de contrato". Pode também editar um tipo de contrato já existente ao escolher (f). C1. Caso decida cancelar a edição do operador, pode voltar à lista de operadores ao clicar em (g).	D1. Se a consistência de dados não for correcta em 4, volta a 1, indicando campos com erro.

Adicionar tipo de contrato

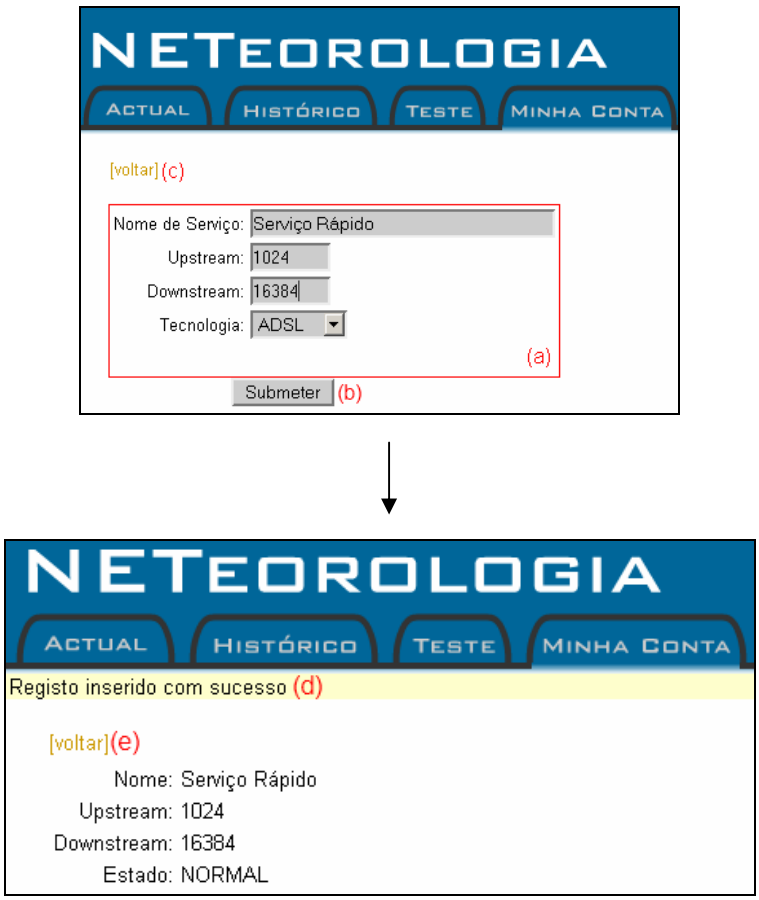


Figura 42 – *Layout* Adicionar tipo de contrato

Nome:	**Adicionar tipo de contrato**	
Âmbito:	NETeorologia	
Finalidade:	Administrador adiciona ao sistema um novo tipo de contrato, para que este possa ser escolhidos pelos utilizadores.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Ao consultar operador, administrador pretende adicionar novo tipo de contrato. Sistema mostra formulário, administrador preenche e submete. Sistema regista novo tipo de contrato.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Mostra formulário em (a), com campos para serem preenchidos (Nome, Velocidade *Upstream*, Velocidade *Downstream*). Todos os campos são obrigatórios.
	2. Preenche nome, velocidade de *upstream*, velocidade de *downstream*. Selecciona tipo de tecnologia.	
	3. Submete dados, clicando em (b).	
		4. Regista novo tipo de contrato, associando-o ao operador.
		5. Mostra mensagem de sucesso da operação em (d), com os dados do novo tipo de contrato. CaU termina.
Sequências alternativas e extensões:	A1. Se não pretender registar o novo tipo de contrato, pode escolher (c), que lhe permite retornar à edição do operador.	
	B1. Pode escolher (e), para voltar à edição do operador.	
		C1. Caso haja algum problema nos dados inseridos, reporta esse erro em (d).

Editar tipo de contrato

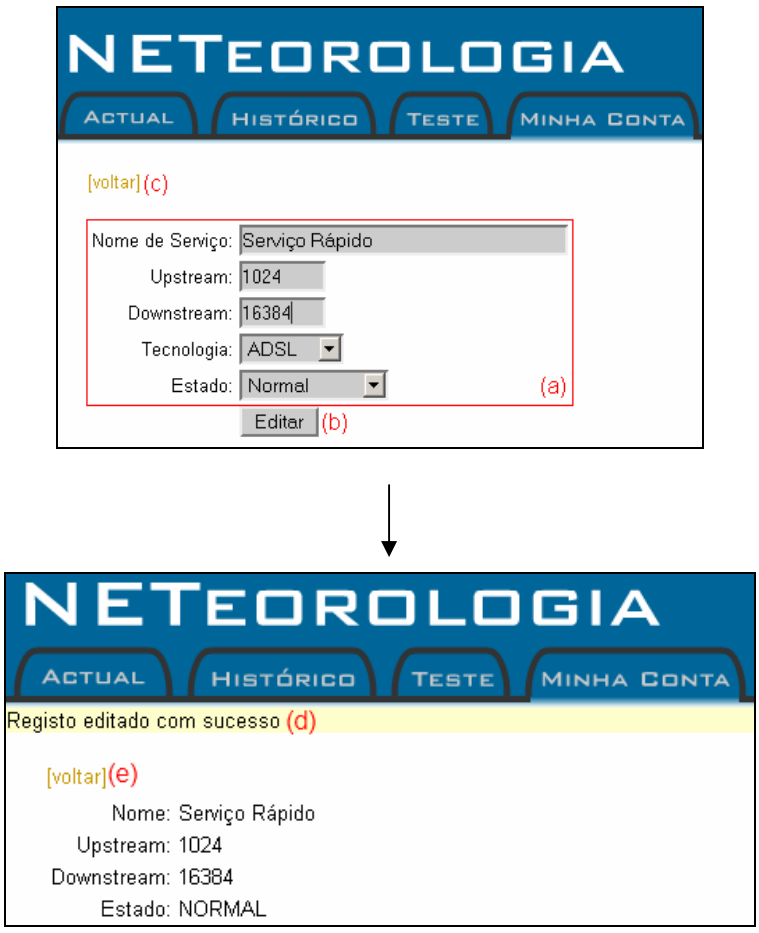


Figura 43 – *Layout* Editar tipo de contrato

Nome:	**Editar tipo de contrato**	
Âmbito:	NETeorologia	
Finalidade:	Administrador altera dados que caracterizam determinado tipo de contrato.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Sistema mostra dados referentes ao contrato. Administrador altera dados que pretende e submete-os. Sistema regista alterações.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Mostra formulário em (a) com campos preenchidos com valores anteriores (Nome, velocidade *upstream*, velocidade *downstream*, tecnologia, estado).
	2. Altera valor dos campos que pretende.	
	3. Submete alterações, clicando em (b).	
		4. Regista alterações.
		5. Mostra mensagem de sucesso da operação, em (d), em conjunto com os dados do operador. CaU termina.
Sequências alternativas e extensões:	A1. Caso pretenda cancelar, pode clicar em (c), o que lhe permite voltar à página anterior do operador.	
	B1. Ao escolher (e), volta à consulta do operador.	
		C1. Caso haja algum problema nos dados inseridos, reporta esse erro em (d).

Adicionar IP Range

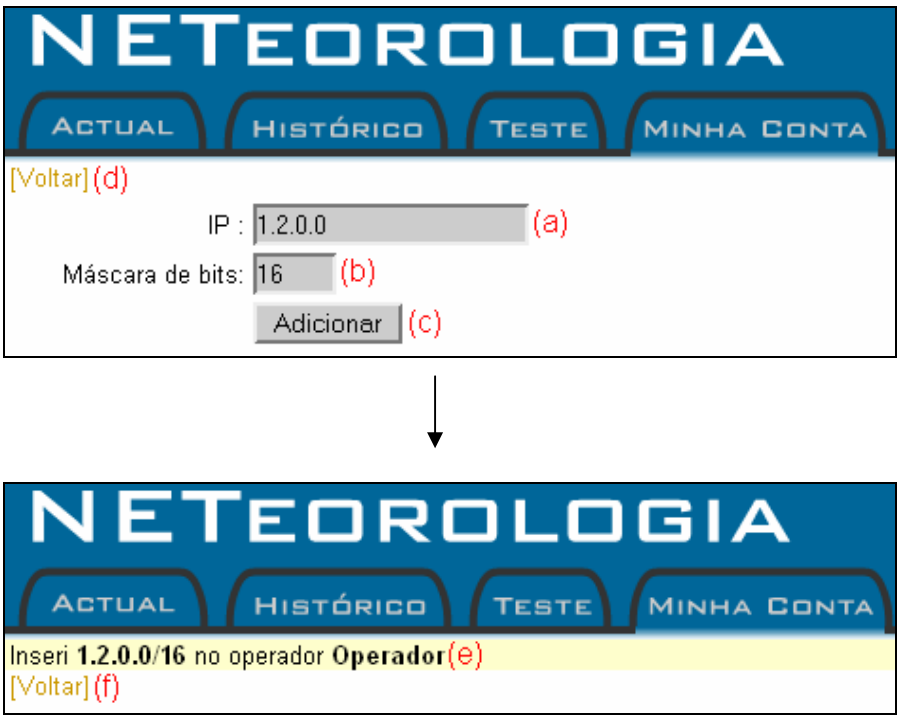


Figura 44 – *Layout* Adicionar IP Range

Nome:	**Adicionar IP Range**	
Âmbito:	NETeorologia	
Finalidade:	Administrador regista novo IP range para operador.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Actor insere valor de endereço de sub rede IP e respectiva máscara de bits, submetendo os dados. Sistema regista novo IP range para operador em questão.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Mostra formulário para preenchimento de valores de endereço de sub rede IP em (a) e máscara de bits em (b).
	2. Preenche campos.	
	3. Submete formulário.	
		4. Verifica dados submetidos.
		5. Regista dados e mostra mensagem de sucesso da operação, em (e). CaU termina.
Sequências alternativas e extensões:	A1. Caso pretenda cancelar, pode clicar em (d), o que lhe permite voltar à página de edição do operador.	B1. Caso encontre erros nos dados, mostra mensagem a indicar o facto.

Remover IP Range

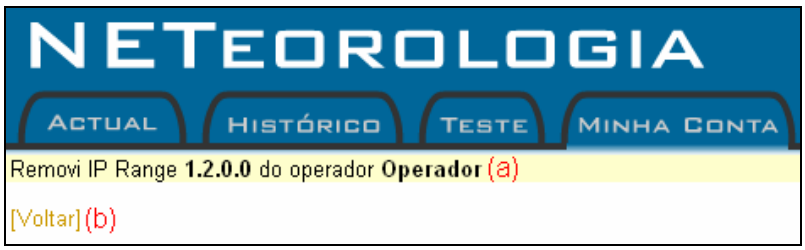


Figura 45 – Remover IP Range

Nome:	**Remover IP Range**	
Âmbito:	NETeorologia	
Finalidade:	Administrador remove IP range de um operador.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Actor insere valor de endereço de sub rede IP e respectiva máscara de bits, submetendo os dados. Sistema regista novo IP range para operador em questão.	
Sequência típica dos eventos:	**Actor** 1. Escolhe IP range a remover.	**Sistema** 2. Mostra página, com mensagem de sucesso da operação em (a). CaU termina.
Sequências alternativas e extensões:	A1. Ao escolher (b), volta à página de edição do operador.	

Listar utilizadores

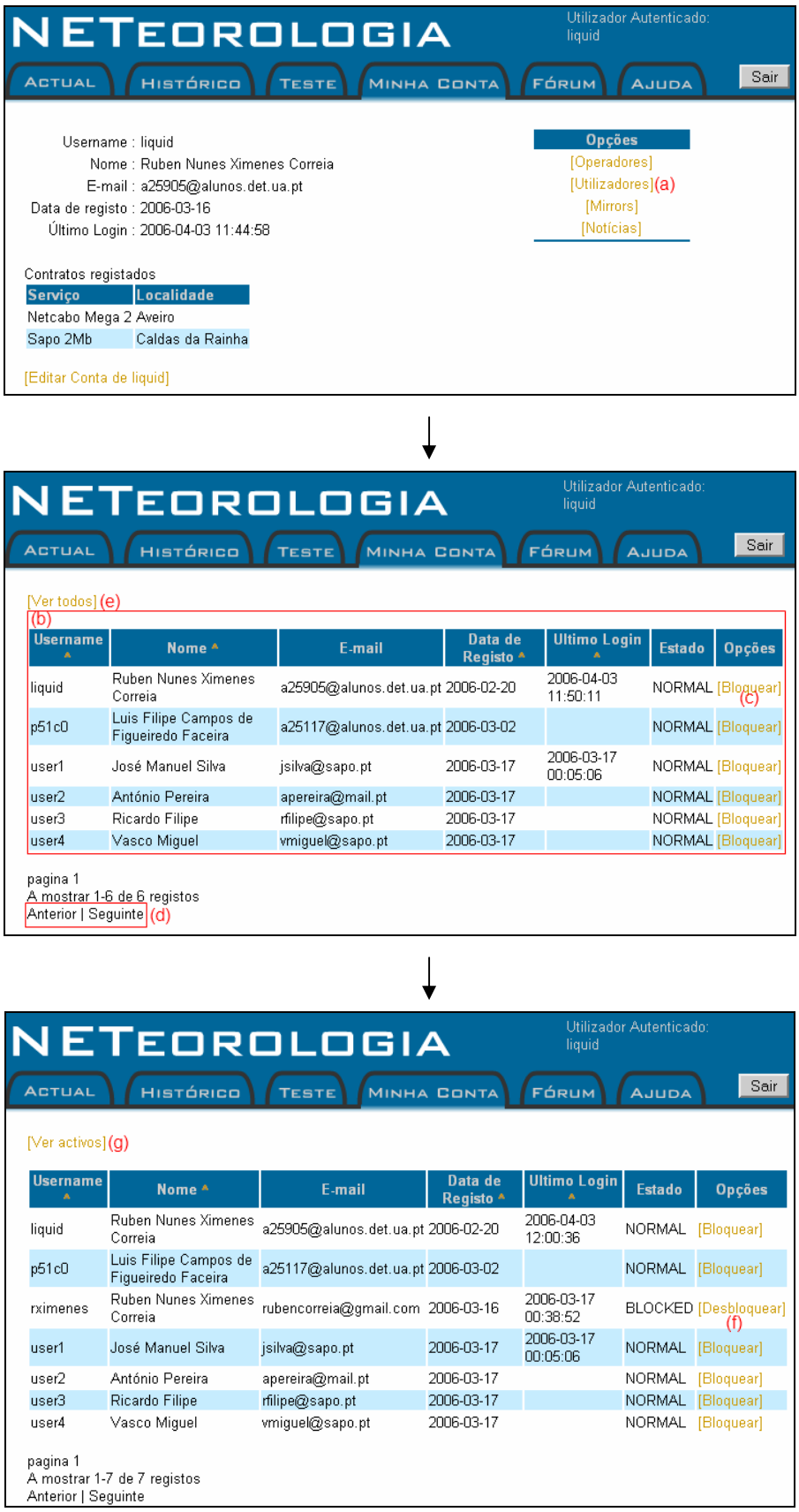


Figura 46 – *Layout* Listar Utilizadores

Nome:	**Listar utilizadores**	
Âmbito:	NETeorologia	
Finalidade:	Sistema mostra lista de utilizadores actualmente registados.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Administrador pede ao sistema para que lhe mostre a lista de utilizadores registados. Sistema devolve lista.	
Sequência típica dos eventos:	**Actor** 1. Actor escolhe (a). 3. Actor consulta informação. CaU termina.	**Sistema** 2. Sistema mostra página com lista dos utilizadores que estão registados no sistema em (b), apresentando os seus dados (nome, *username*, *e-mail*, data de registo, último *login*, estado).
Sequências alternativas e extensões:	A1. Actor pode escolher (c), caso o utilizador esteja no estado normal, para iniciar o CaU que permite bloquear o utilizador em causa, "Bloquear utilizador". B1. Caso hajam mais páginas anteriormente ou posteriormente, pode navegar para essas páginas através das opções apresentadas em (d). C1. Se pretender ver todos os utilizadores (normais e bloqueados), pode escolher (e). D1. Actor pode escolher (f), caso o utilizador esteja no estado bloqueado, para iniciar o CaU "Desbloquear utilizador". E1. Se pretender ver a lista apenas com utilizadores normais, escolhe (g). F1. Pode ordenar os campos de *username*, nome, data de registo e data de último *login*, de forma ascendente e descendente, através da opção existente no cabeçalho da tabela.	

Bloquear utilizador

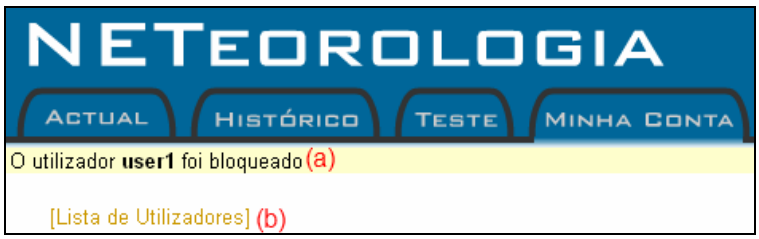


Figura 47 – *Layout* Bloquear utilizador

Nome:	**Bloquear utilizador**	
Âmbito:	NETeorologia	
Finalidade:	Administrador bloqueia a conta de determinado utilizador.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Administrador escolhe utilizador a bloquear, dentro da lista de utilizadores. Sistema mostra mensagem a indicar que utilizador foi bloqueado.	
Sequência típica dos eventos:	**Actor**	**Sistema** 1. Indica sucesso da operação em (a), apresentando o *username* do utilizador bloqueado. CaU termina.
Sequências alternativas e extensões:	A1. Se pretender, pode escolher (b).	A2. Volta a lista de utilizadores.

Desbloquear utilizador

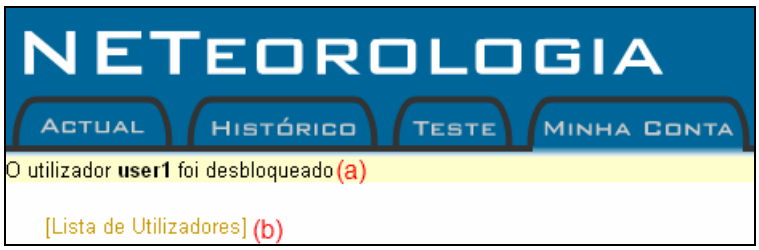


Figura 48 – *Layout* Desbloquear utilizador

Nome:	**Desbloquear utilizador**	
Âmbito:	NETeorologia	
Finalidade:	Administrador desbloqueia a conta de determinado utilizador.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Administrador escolhe utilizador a desbloquear, dentro da lista de utilizadores. Sistema mostra mensagem a indicar que utilizador foi desbloqueado.	
Sequência típica dos eventos:	**Actor**	**Sistema** 1. Indica sucesso da operação em (a), apresentando o *username* do utilizador bloqueado. CaU termina.
Sequências alternativas e extensões:	A1. Se pretender, pode escolher (b).	A2. Volta a lista de utilizadores.

Listar *Mirrors*

Figura 49 – *Layout* Listar *Mirrors*

Nome:	**Listar *mirrors***	
Âmbito:	NETeorologia	
Finalidade:	Administrador consulta lista com informação referente aos dados dos diversos *mirrors* registados no sistema.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Actor pede ao sistema para que lhe mostre a lista de *mirrors* existentes. Sistema devolve lista.	
Sequência típica dos eventos:	**Actor** 1. Escolhe a opção (a).	**Sistema** 2. Mostra página de teste. Apresenta a lista de *mirrors* activos em (b). CaU termina.
Sequências alternativas e extensões:	A1. Se pretender, pode escolher (c). Desta forma, inicia o CaU "Realizar medição *online*". B1. Se escolher (d), pode iniciar o CaU "Editar *Mirror*". C1. Pode adicionar um novo *mirror*, escolhendo a opção (e), iniciando o CaU "Adicionar *Mirror*". D1. Escolhe (f).	D2. Sistema mostra lista com todos os *mirrors* (activos e desactivados).

Adicionar *Mirror*

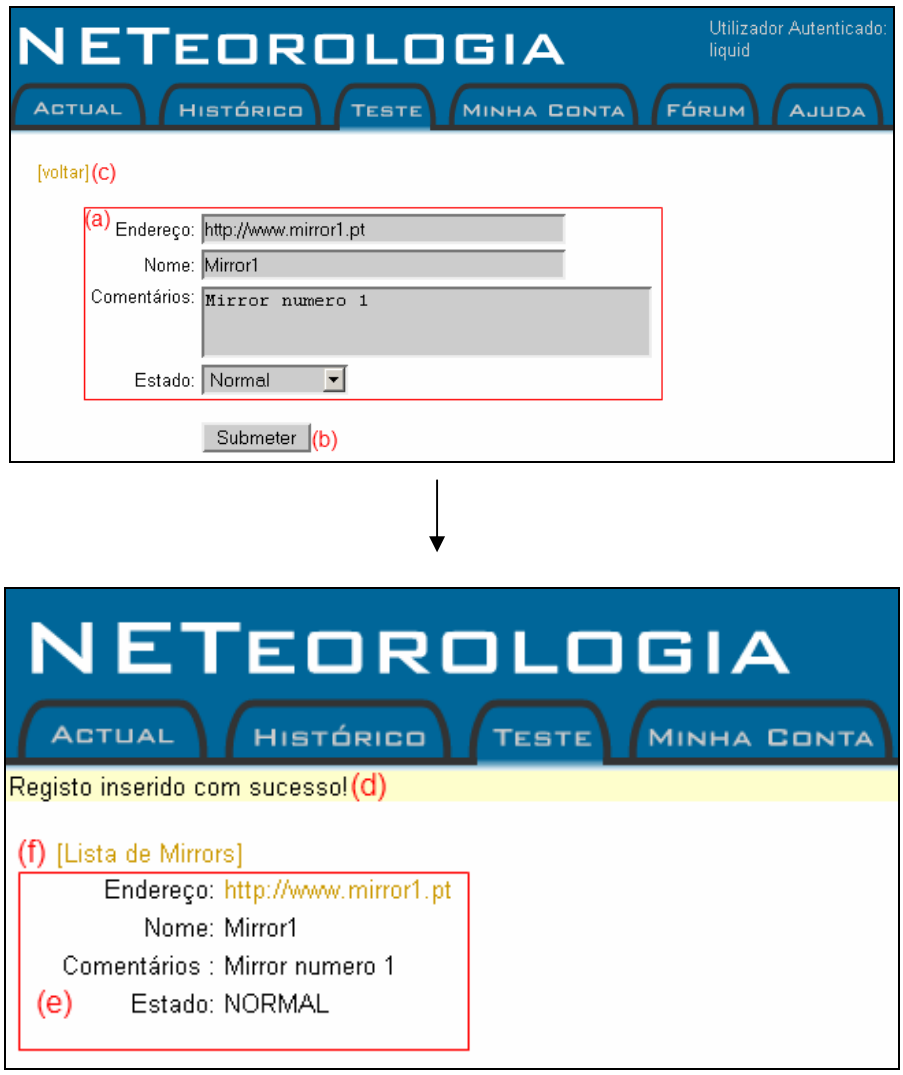


Figura 50 – *Layout* Adicionar *mirror*

Nome:	**Adicionar *mirror***	
Âmbito:	NETeorologia	
Finalidade:	O administrador adiciona ao sistema informação referente a um novo *mirror*.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado.	
Requisitos funcionais:	Administrador indica dados de novo operador. Sistema guarda registo desses dados.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Sistema mostra ecrã para inserção de dados relativos ao novo *mirror*: endereço, nome (obrigatórios ambos), comentário e estado em (a)
	2. Insere dados referentes ao novo *mirror*.	
	3. Submete dados, clicando em (b).	
		4. Verifica consistência dos dados.
		5. Regista novo *mirror* na base de dados.
		6. Mostra mensagem de sucesso da operação em (d), apresentando igualmente os dados registados para aquele *mirror* em (e).
Sequências alternativas e extensões:		A1. A verificação encontra erros nos dados que foram submetidos. Volta ao passo 1, indicando erros que ocorreram.
	B1. Ao clicar em (c) ou em (f), volta à lista de *mirrors*.	

Editar *Mirror*

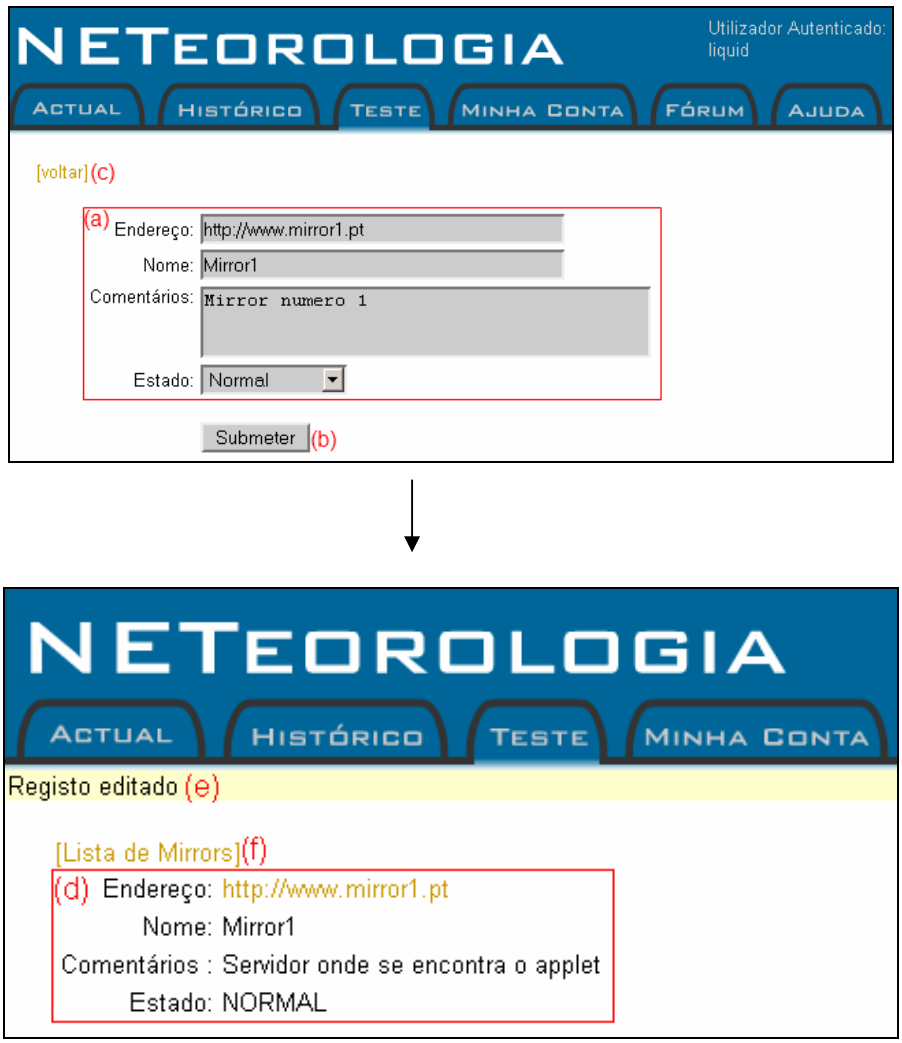


Figura 51 – Editar *mirror*

Nome:	**Editar *mirror***	
Âmbito:	NETeorologia	
Finalidade:	O administrador edita dados de *mirror*.	
Actores:	Administrador	
Pré-condições:	Administrador foi correctamente identificado, *mirror* registado no sistema.	
Requisitos funcionais:	Sistema mostra formulário preenchido com dados do *mirror*. Actor editar dados que pretende e submete. Sistema verifica dados e guarda as alterações.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Sistema mostra ecrã com dados do *mirror* (endereço, nome, comentário e estado), em (a)
	2. Edita dados do *mirror*.	
	3. Submete dados, clicando em (b).	
		4. Verifica consistência dos dados.
		5. Regista alterações.
		6. Mostra mensagem de sucesso da operação em (d), apresentando igualmente os dados editados do *mirror* em (e).
Sequências alternativas e extensões:		A1. A verificação encontra erros nos dados que foram submetidos. Volta ao passo 1, indicando erros que ocorreram.
	B1. Ao clicar em (c) ou (f), volta à lista de *mirrors*.	

Listar Notícias

Figura 52 – *Layout* Listar Notícias

Nome:	**Listar notícias**	
Âmbito:	NETeorologia	
Finalidade:	Administrador consulta lista das notícias registadas no sistema	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Actor pede ao sistema para que lhe mostre a lista de notícias existentes. Sistema devolve lista.	
Sequência típica dos eventos:	**Actor** 1. Escolhe a opção (a).	**Sistema** 2. Mostra lista de notícias registadas no sistema em (b). CaU termina.
Sequências alternativas e extensões:	A1. Ao escolher (c), inicia o CaU “Consultar notícia”. B1. Ao escolher (d), inicia o CaU “Editar Notícia”. C1. Ao escolher (e), inicia o CaU “Remover Notícia”. D1. Se pretender adicionar uma nova notícia, pode fazê-lo ao escolher (f). E1. Se escolher (g), volta à página inicial.	

Adicionar Notícia

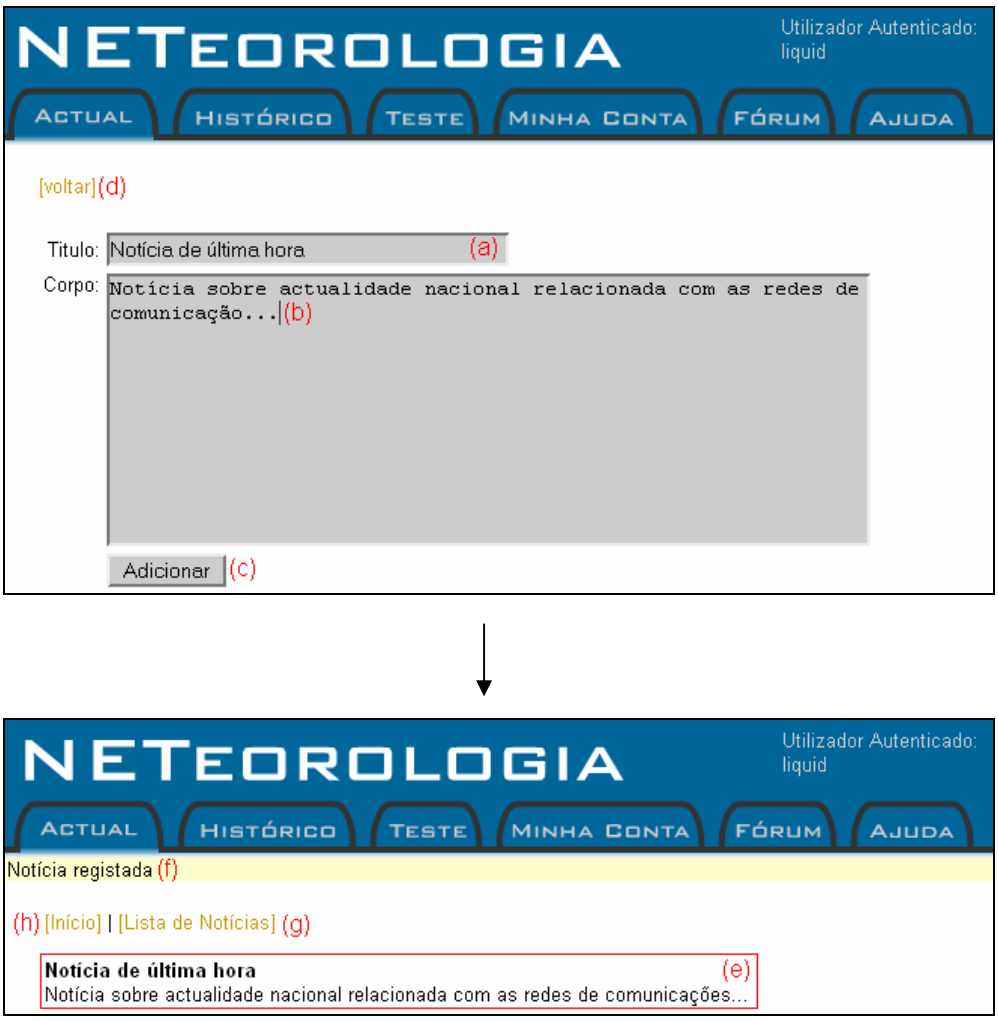


Figura 53 – *Layout* Adicionar Notícia

Nome:	**Adicionar notícia**	
Âmbito:	NETeorologia	
Finalidade:	Administrador regista nova notícia no sistema.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Sistema mostra formulário para inserção de notícia. Actor insere título e corpo de notícia, submetendo em seguida. Sistema regista dados.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Mostra formulário de inserção de notícia.
	2. Insere título em (a) e notícia em (b).	
	3. Submete formulário, clicando em (c).	
		4. Verifica que campos foram preenchidos.
		5. Regista nova notícia, mostrando uma mensagem de sucesso em (f) e apresentando o texto da notícia em (e). CaU termina.
Sequências alternativas e extensões:	A1. Ao escolher (h) ou (g), volta à página inicial ou à lista de notícias, respectivamente. B1. Caso pretenda cancelar, pode escolher (d) para voltar à lista de notícias.	

Consultar Notícia

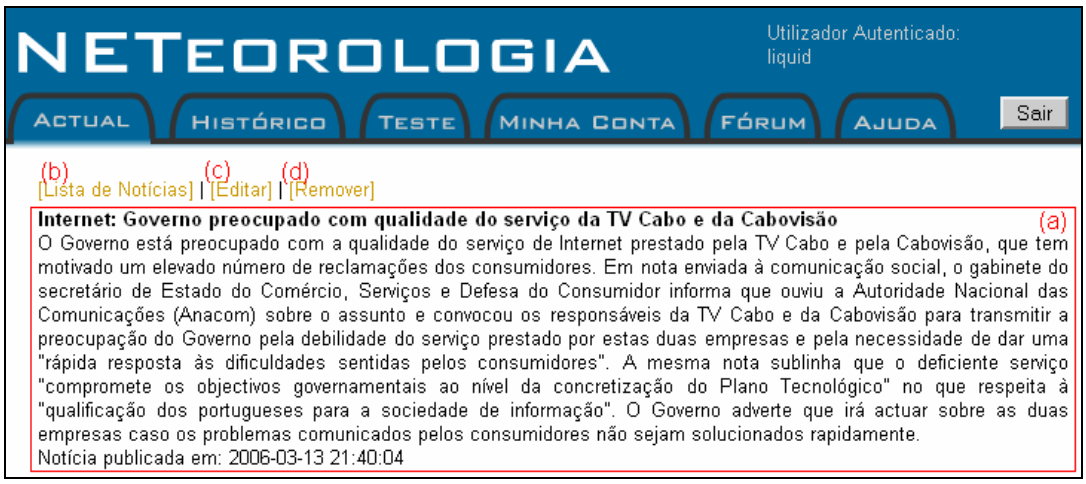


Figura 54 – *Layout* Consultar Notícia

Nome:	**Consultar notícia**	
Âmbito:	NETeorologia	
Finalidade:	Administrador consulta notícia.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Sistema mostra notícia (data, título e corpo). Actor consulta a notícia.	
Sequência típica dos eventos:	**Actor** 2. Consulta notícia. CaU termina.	**Sistema** 1. Mostra notícia em (a).
Sequências alternativas e extensões:	A1. Se pretender, pode escolher (b). Desta forma volta a lista de notícias. B1. Se escolher (c), inicia o CaU “Editar Notícia”. C1. Para remover a notícia, pode escolher (d).	

Editar Notícia

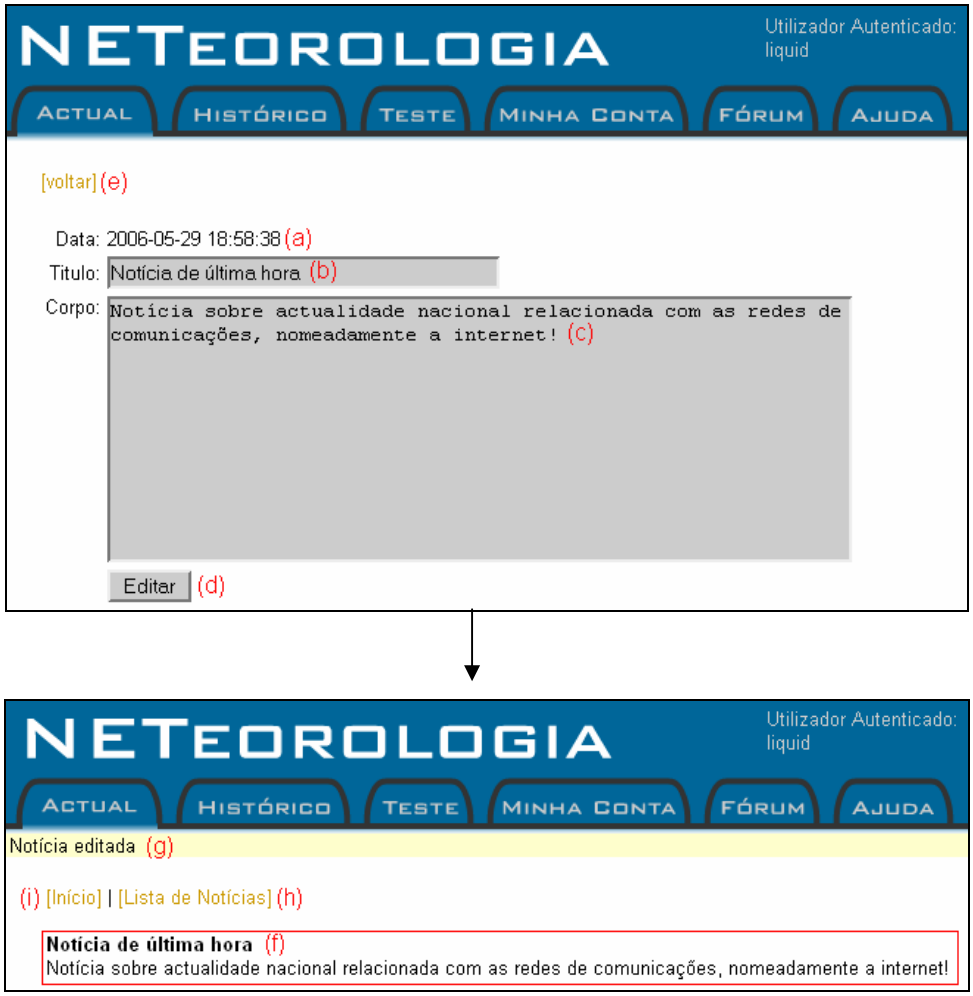


Figura 55 – *Layout* Editar Notícia

Nome:	**Editar notícia**	
Âmbito:	NETeorologia	
Finalidade:	Administrador edita notícia existente no sistema.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Sistema mostra formulário para edição de notícia. Actor edita campos que pretende e submete. Sistema regista dados.	
Sequência típica dos eventos:	**Actor**	**Sistema**
		1. Mostra formulário com campos preenchidos com data em (a), título em (b) e corpo da notícia em (c).
	2. Edita título e/ou corpo da notícia.	
	3. Submete formulário, clicando em (d).	
		4. Verifica que campos foram preenchidos.
		5. Regista alterações à notícia, mostrando uma mensagem de sucesso em (g) e apresentando o texto da notícia em (f). CaU termina.
Sequências alternativas e extensões:	A1. Ao escolher (i) ou (h), volta à página inicial ou à lista de notícias, respectivamente. B1. Caso pretenda cancelar, pode escolher (e) para voltar à lista de notícias.	

Remover Notícia

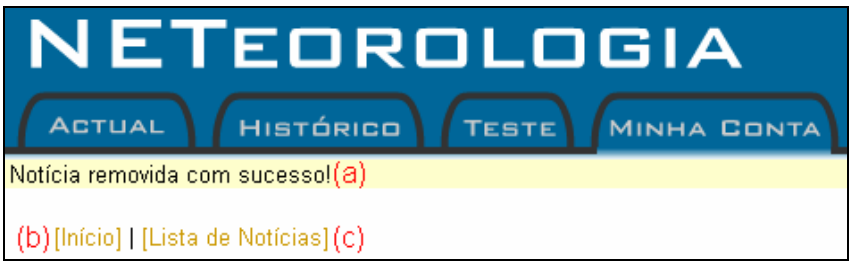


Figura 56 – *Layout* Remover Notícia

Nome:	**Remover notícia**	
Âmbito:	NETeorologia	
Finalidade:	Administrador remove notícia existente no sistema.	
Actores:	Administrador	
Tipo:	Real	
Descrição geral:	Actor indica notícia a remover. Sistema remove notícia.	
Sequência típica dos eventos:	**Actor** 1. Escolhe notícia que pretende remover.	**Sistema** 2. Mostra página, com mensagem de sucesso da operação em (a). CaU termina.
Sequências alternativas e extensões:	A1. Ao escolher (b) ou (c), volta à página inicial ou à lista de notícias, respectivamente.	